HEWLETT PACKARD

HP 9000
Computers

HP-UX Reference
Volume 3

HP-UX Reference

Volume 3: Sections 1M, 4, 5, and 7

HP 9000 Computers

HP-UX Release 9.0

HP Part No. B2355-90033
Printed in USA August 1992

Third Edition
E0892

Legal Notices

The information contained in this document is subject to change without notice.

Hewlett-Packard Company makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard Company shall not be liable for errors contained herein or direct, indirect, special, incidental, or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty: A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Printing History

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. However, minor changes may be made at reprint without changing the printing date. The manual part number changes when extensive changes are made.

To ensure that you receive new editions of this manual when changes occur, you may subscribe to the appropriate product support service, available through your HP sales representative.

August 1992. Third Edition. This edition is an update to the Second Edition and is valid for HP-UX Release 9.0 on all HP 9000 systems. Replaces Second Edition, HP part number B2355-90004.

June 1991. Second Edition. Update to the First Edition for HP-UX Release 8.05 on Series 700 systems. Also valid for HP-UX Release 8.0 on Series 300/400 and Series 800 systems. Replaces First Edition, HP part number B1864-90000.

January 1991. First Edition. Replaces manual part number 09000-90013. Valid for HP-UX Release 8.0 on Series 300/400, 700, and Series 800 systems. The *Networking Reference* was merged into this manual at Release 8.0.

New Features

This edition contains several new features.

Typography has been changed to conform to style used in other HP manuals as well as industry standards (conversion complete execpt for parts of Volume 3). Command names, argument names, and such appear on the printed page in exactly the same form as when they are typed in commands or applications, eliminating much confusion regarding capitalization of letters, which items are literals or otherwise, etc.

Progressive bleed tabs in each section are positioned vertically on the page edge according to the first letter in the name of the manual entry for easier access.

As part of an on-going effort to improve the quality and usability of this manual, several entries have been expanded and rewritten for better clarity and many examples have been added or expanded in many entries. Many changes are a direct result of comments, requests, and suggestions from users outside of HP.

Manual is expanded considerably to conver new functionality from Open Software Foundation and several other sources as well as newer versions of NFS Services and other software contained in previous releases.

Do You Have Comments or Suggestions?

Comments and suggestions from users about this manual are always welcome because they are an important part of our on-going process of improving the *HP-UX Reference*.

Internal HP users send electronic mail to:

```
hpuxref@fc.hp.com
```

Other users, please use the reply card provided in the manual or send a note or letter by ordinary mail to:

HP-UX Reference Comments, MS 11
Hewlett-Packard Company
3404 East Harmony Road
Fort Collins, CO 80525-9988, U.S.A.

Notes

Table of Contents for Volume 3

Table of Contents
Volume 3

Section 1M: System Administration Commands

Table of Contents
Volume 3

Table of Contents
Volume 3

Table of Contents
Volume 3

Section 4: File Formats

Table of Contents
Volume 3

Section 5: Miscellaneous

Section 7: Device (Special) Files

Table of Contents
Volume 3

Section 1M:
System Administration Commands

NAME

intro - introduction to system maintenance commands and application programs

DESCRIPTION

This section describes commands that are used chiefly for system maintenance and administration purposes. The commands in this section should be used in conjunction with other sections of this manual, as well as the HP-UX System Administration manuals for your system.

Command Syntax

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option* (*s*)] [*cmd_arg* (*s*)]

where the elements are defined as follows:

name Name of an executable file.

option One or more *option*s can appear on a command line. Each takes one of the following forms:

-no_arg_letter
A single letter representing an option without an argument.

-no_arg_letters
Two or more single-letter options combined into a single command-line argument.

-arg_letter<>*opt_arg*
A single-letter option followed by a required argument where:

arg_letter
is the single letter representing an option that requires an argument,

opt_arg
is an argument (character string) satisfying the preceding *arg_letter*,

<> represents optional white space.

cmd_arg Path name (or other command argument) *not* beginning with -, or - by itself indicating the standard input. If two or more *cmd_arg*s appear, they must be separated by white space.

RETURN STATUS

Upon termination, each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (for descriptions, see *wait*(2) and *exit*(2)). The system-supplied byte is 0 for normal termination. The byte provided by the program is customarily 0 for successful execution and non-zero to indicate errors or failure such as incorrect parameters in the command line, or bad or inaccessible data. Values returned are usually called variously "exit code", "exit status", or "return code", and are described only where special conventions are involved.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.

SEE ALSO

getopt(1), exit(2), wait(2), getopt(3C), hier(5).

The introduction to this manual.

NAME

accept, reject - allow/prevent LP requests

SYNOPSIS

`/usr/lib/accept` *destinations*
`/usr/lib/reject` [`-r`[*reason*]] *destinations*

DESCRIPTION

`accept` allows `lp` to accept requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use `lpstat` to find the status of *destinations*.

`reject` prevents `lp` from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use `lpstat` to find the status of *destinations*.

Options

The following option is useful with `reject`:

`-r`[*reason*] Associates a *reason* with preventing `lp` from accepting requests. This *reason* applies to all printers mentioned up to the next `-r` option. *reason* is reported by `lp` when users direct requests to the named *destinations* and by `lpstat`. If the `-r` option is not specified or is specified without a *reason*, a default *reason* is used. The maximum length of the *reason* message is 80 bytes.

HP Clustered Environment

In the HP Clustered Environment, all spooling is handled as if the cluster nodes were a single system and all printers attached to either the cluster server or clients can be available. Remote spooling applies to spooling from or to machines outside of the cluster nodes.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, *accept* and *reject* behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte code sets are supported.

WARNINGS

`accept` and `reject` perform their operation on the local system (or HP cluster) only.

FILES

`/usr/spool/lp/*`

SEE ALSO

enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1), rcancel(1M), rlp(1M), rlpdaemon(1M), rlpstat(1M).

NAME

acctdisk, acctdusg, accton, acctwtmp - overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk

/usr/lib/acct/acctdusg [**-u** *file*] [**-p** *file*]

/usr/lib/acct/accton [*file*]

/usr/lib/acct/acctwtmp *reason*

DESCRIPTION

Accounting software is structured as a set of tools (consisting of both C programs and shell procedures) that can be used to build accounting systems. The shell procedures, described in *acctsh*(1M), are built on top of the C programs.

Connect time accounting is handled by various programs that write records into **/etc/utmp**, as described in *utmp*(4). The programs described in *acctcon*(1M) convert this file into session and charging records which are then summarized by *acctmerg*(1M).

Process accounting is performed by the HP-UX system kernel. Upon termination of a process, one record per process is written to a file (normally **/usr/adm/pacct**). The programs in *acctprc*(1M) summarize this data for charging purposes; *acctcms*(1M) is used to summarize command usage. Current process data can be examined using *acctcom*(1M).

Process accounting and connect time accounting (or any accounting records in the format described in *acct*(4)) can be merged and summarized into total accounting records by *acctmerg* (see **tacct** format in *acct*(4)). *prtacct* (see *acctsh*(1M)) is used to format any or all accounting records.

acctdisk reads lines that contain user ID, login name, and number of disk blocks, and converts them to total accounting records that can be merged with other accounting records.

acctdusg reads its standard input (usually from **find / -print**) and computes disk resource consumption (including indirect blocks) by login. Only files found under login directories (as determined from the password file) are accounted for. All files under a login directory are assumed to belong to that user regardless of actual owner. If **-u** is given, records consisting of those file names for which *acctdusg* charges no one are placed in *file* (a potential source for finding users trying to avoid disk charges). If **-p** is given, *file* is the name of the password file. This option is not needed if the password file is **/etc/passwd**. (See *diskusg*(1M) for more details.)

accton turns process accounting off if the optional *file* argument is omitted. If *file* is given, it must be the name of an existing file, to which the kernel appends process accounting records (see *acct*(2) and *acct*(4)).

acctwtmp writes a *utmp*(4) record to its standard output. The record contains the current time and a string of characters that describe the *reason* for writing the record. A record type of ACCOUNTING is assigned (see *utmp*(4)). The string argument *reason* must be 11 or fewer characters, numbers, **$**, or spaces. For example, the following are suggestions for use in reboot and shutdown procedures, respectively:

```
acctwtmp 'uname' >> /etc/wtmp
acctwtmp "file save" >> /etc/wtmp
```

In the HP Clustered environment, accounting software collects data on a per-machine basis. Accounting data for the entire cluster can be merged using the *acctmerg*(1M) command.

FILES

/usr/lib/acct	holds all accounting commands listed in section (1M) of this manual
/usr/adm/pacct	current process accounting file
/etc/passwd	used for login name to user ID conversions
/etc/wtmp	login/logoff history file

SEE ALSO

acctcms(1M), acctcom(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), diskusg(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4),

System Accounting topics in *HP-UX System Administrator* manuals.

STANDARDS CONFORMANCE

acctdisk: SVID2

accton: SVID2

acctwtmp: SVID2

NAME

acctcms - command summary from per-process accounting records

SYNOPSIS

`/usr/lib/acct/acctcms` [*options*] *files*

DESCRIPTION

`acctcms` reads one or more *files*, normally in the form described in *acct*(4). It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format.

Options

`acctcms` recognizes the following options:

`-a` Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, "hog factor", characters transferred, and blocks read and written, as in *acctcom*(1M). Output is normally sorted by total kcore-minutes.

`-c` Sort by total CPU time, rather than total kcore-minutes.

`-j` Combine all commands invoked only once under `***other`.

`-n` Sort by number of command invocations.

`-s` Any file names encountered hereafter are already in internal summary format.

`-t` Process all records as total accounting records. The default internal summary format splits each field into prime- and non-prime-time parts. This option combines the prime and non-prime time parts into a single field that is the total of both, and provides upward compatibility with old (i.e., UNIX System V) style `acctcms` internal summary format records.

The following options can be used only with the `-a` option.

`-p` Output a prime-time-only command summary.

`-o` Output a non-prime- (offshift) time only command summary.

When `-p` and `-o` are used together, a combination prime and non-prime time report is produced. All the output summaries are total usage except number of times executed, CPU minutes, and real minutes which are split into prime and non-prime.

EXAMPLES

A typical sequence for performing daily command accounting and for maintaining a running total is:

```
acctcms ile ... >today
cp total previoustotal
acctcms -s today previoustotal >total
acctcms -a -s today
```

SEE ALSO

acct(1M), acctcom(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

WARNINGS

Unpredictable output results if `-t` is used on new-style internal-summary-format files, or if it is not used with old style internal summary format files.

STANDARDS CONFORMANCE

`acctcms`: SVID2

NAME

acctcom - search and print process accounting file(s)

SYNOPSIS

/usr/lib/acct/acctcom [[options] [file]] . . .

DESCRIPTION

acctcom reads *file*, the standard input, or **/usr/adm/pacct**, in the form described by *acct*(4) and writes selected records to the standard output. Each record represents the execution of one process. The output shows the **COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SECS), CPU (SECS), MEAN SIZE(K)**, and optionally, **F** (the *fork/exec* flag: **1** for *fork* without *exec*), **STAT** (the system exit status), **HOG FACTOR, KCORE MIN, CPU FACTOR, CHARS TRNSFD**, and **BLOCKS READ** (total blocks read and written).

The command name is preceded by a **#** if it was executed with *super-user* privileges. If a process is not associated with a known terminal, a **?** is printed in the **TTYNAME** field.

The system exit status is **0** if the process terminated by calling *exit*. If it is not **0**, it is the signal number that caused the process to terminate. If a core file image was produced as a result of the signal (see *signal*(5)), it is the signal number plus **0200**.

If no *files* are specified, and if the standard input is associated with a terminal or **/dev/null** (as is the case when using **&** in the shell), **/usr/adm/pacct** is read; otherwise, the standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward; i.e., in chronological order by process-completion time. The file **/usr/adm/pacct** is usually the current file to be examined; a busy system may need several such files of which all but the current file are found in **/usr/adm/pacct***?*. The *options* are:

-a Show some average statistics about the processes selected. Statistics are printed after the output records.

-b Read backwards, showing latest commands first. This *option* has no effect when the standard input is read.

-f Print in octal the *fork/exec* flag and system exit status columns in the output.

-h Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:

$$\frac{\textit{total CPU time}}{\textit{elapsed time}}$$

-i Print columns containing the I/O counts in the output.

-k Instead of memory size, show total kcore-minutes.

-m Show mean core size (the default).

-r Show CPU factor (user time/(system-time + user-time).

-t Show separate system and user CPU times.

-v Exclude column headings from the output.

-l *line* Show only processes belonging to terminal **/dev/***line*.

-u *user* Show only processes belonging to *user* that can be specified by: a user ID, a login name that is then converted to a user ID, a **#** which designates only those processes executed with *super-user* privileges, or **?** which designates only those processes associated with unknown user IDs.

-g *group* Show only processes belonging to *group*. The *group* can be designated by either the group ID or group name.

-s *time* Select processes existing at or after *time*, given in the format *hr* [*:min* [*:sec*]].

-e *time* Select processes existing at or before *time*. Using the same *time* for both **-s** and **-e** shows the processes that existed at *time*.

-S *time* Select processes starting at or after *time*.

-E *time* Select processes ending at or before *time*.

-n *pattern* Show only commands matching *pattern* where *pattern* is a regular expression as in *ed*(1) except that + means one or more occurrences.

-q Do not print any output records. Just print the average statistics as with the **-a** option.

-o *ofile* Copy selected process records in the input data format to *ofile*. Supress standard output printing.

-H *factor* Show only processes that exceed *factor*, where *factor* is the "hog factor" as explained in option **-h** above.

-O *time* Show only those processes with operating system CPU time exceeding *time*.

-C *sec* Show only processes with total CPU time, system plus user, exceeding *sec* seconds.

-I *chars* Show only processes transferring more characters than the cut-off number given by *chars*.

Listing options together has the effect of a logical AND.

FILES

/etc/group
/usr/adm/pacct
/etc/passwd

SEE ALSO

ps(1), su(1), acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), wait(2), acct(4), utmp(4), signal(5).

BUGS

acctcom only reports on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time, *time* is interpreted as occurring on the previous day.

STANDARDS CONFORMANCE

acctcom: SVID2

NAME

acctcon1, acctcon2 - connect-time accounting

SYNOPSIS

`/usr/lib/acct/acctcon1` [*options*]

`/usr/lib/acct/acctcon2`

DESCRIPTION

`acctcon1` converts a sequence of login/logoff records read from its standard input to a sequence of records, one per login session. Its input should normally be redirected from `/etc/wtmp`. Its output is ASCII, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time. Prime connect time is defined as the connect time within a specific prime period on a non-holiday weekday (Monday through Friday). The starting and ending time of the prime period and the year's holidays are defined in file `/usr/lib/acct/holidays`.

`acctcon1` recognizes the following *options*:

`-p` Print input only, showing line name, login name, and time (in both numeric and date/time formats).

`-t` `acctcon1` maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The `-t` flag causes it to use, instead, the last time found in its input, thus ensuring reasonable and repeatable numbers for non-current files.

`-l` *file* *file* is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Hang-up, termination of `login` (see *login*(1)), and termination of the login shell each generate logoff records, so that the number of logoffs is often three to four times the number of sessions. See *init*(1M) and *utmp*(4).

`-o` *file* *file* is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

acctcon2 expects as input a sequence of login session records and converts them into total accounting records (see `tacct` format in *acct*(4)).

EXAMPLES

These commands are typically used as shown below. The file `ctmp` is created only for the use of commands described by the *acctprc*(1M) manual entry:

```
acctcon1 -t -l lineuse -o reboots <wtmp | sort +1n +2 >ctmp
acctcon2 <ctmp | acctmerg >ctacct
```

FILES

`/etc/wtmp`
`/usr/lib/acct/holidays`

WARNINGS

The line usage report is confused by date changes. Use `wtmpfix` (see *fwtmp*(1M)) to correct this situation.

SEE ALSO

acct(1M), acctcms(1M), acctcom(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), init(1M), login(1), runacct(1M), acct(2), acct(4), utmp(4).

STANDARDS CONFORMANCE

`acctcon1`: SVID2

`acctcon2`: SVID2

NAME

acctmerg - merge or add total accounting files

SYNOPSIS

/usr/lib/acct/acctmerg [*options*] [*file*] ...

DESCRIPTION

acctmerg reads its standard input and up to nine additional files, all in the **tacct** format (see *acct*(4)) or an ASCII version thereof. It merges these inputs by adding records whose keys (normally user ID and name) are identical, and expects the inputs to be sorted on those keys.

Options

acctmerg recognizes the following options:

- **-a** Produce output in ASCII version of **tacct**.
- **-i** Input files are in ASCII version of **tacct**.
- **-p** Print input with no processing.
- **-t** Produce a single record that totals all input.
- **-u** Summarize by user ID, rather than user ID and name.
- **-v** Produce output in verbose ASCII format, with more precise notation for floating point numbers.

EXAMPLES

The following sequence is useful for making "repairs" to any file kept in this format:

```
acctmerg -v <file1 >file2
      edit file2 as desired ...
acctmerg -i <file2 >file1
```

SEE ALSO

acct(1M), acctcms(1M), acctcom(1M), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

STANDARDS CONFORMANCE

acctmerg: SVID2

NAME

acctprc1, acctprc2 - process accounting

SYNOPSIS

`/usr/lib/acct/acctprc1 [ ctmp ]`

`/usr/lib/acct/acctprc2`

DESCRIPTION

`acctprc1` reads input in the form described by *acct*(4), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in memory segment units). If `ctmp` is given, it is expected to contain a list of login sessions in the form described in *acctcon*(1M), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in `ctmp` helps it distinguish among different login names that share the same user ID.

`acctprc2` reads records in the form written by `acctprc1`, summarizes them by user ID and name, then writes the sorted summaries to the standard output as total accounting records.

These commands are typically used as shown below:

```
acctprc1 ctmp </usr/adm/pacct | acctprc2 >ptacct
```

EXTERNAL INFLUENCES

Environment Variables

For the output of `acctprc2`, if the user IDs are identical, `LC_COLLATE` determines the order in which the user names are sorted.

If `LC_COLLATE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `acctprc2` behaves as if all internationalization variables are set to "C" (see *environ*(5)).

FILES

`/etc/passwd`

SEE ALSO

acct(1M), acctcms(1M), acctcom(1M), acctcon(1M), acctmerg(1M), acctsh(1M), cron(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

WARNINGS

Although it is possible to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from `cron` for example (see *cron*(1M)). More precise conversion can be done by faking login sessions on the console via the `acctwtmp` program in *acct*(1M).

A memory segment of the mean memory size is a unit of measure for the number of bytes in a logical memory segment on a particular processor.

STANDARDS CONFORMANCE

`acctprc1`: SVID2

`acctprc2`: SVID2

NAME

chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, shutacct, startup, turnacct - shell procedures for accounting

SYNOPSIS

/usr/lib/acct/chargefee *login-name number*

/usr/lib/acct/ckpacct [*blocks*]

/usr/lib/acct/dodisk [**-o**] [*files ...]*

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct *number*

/usr/lib/acct/nulladm *file*

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily [**-l**] [**-c**] [*-mmdd*]

/usr/lib/acct/prtacct *file* [*heading* [**-l**] [**-c**] [*-mmdd*] [**-l**] [**-c**] [*-mmdd*]]

/usr/lib/acct/shutacct [*reason*]

/usr/lib/acct/startup

/usr/lib/acct/turnacct on | **off** | **switch**

DESCRIPTION

chargefee can be invoked to charge a *number* of units to *login-name*. A record is written to **/usr/adm/fee**, to be merged with other accounting records during the night.

ckpacct should be initiated via *cron*(1M). It periodically checks the size of **/usr/adm/pacct**. If the size exceeds *blocks*, 1000 by default, *turnacct* is invoked with argument *switch*. If the number of free disk blocks in the **/usr** file system falls below 500, *ckpacct* automatically turns off the collection of process accounting records via the **off** argument to *turnacct*. When at least this number of blocks is restored, the accounting will be activated again. This feature is sensitive to the frequency at which *ckpacct* is executed, usually by *cron*.

dodisk should be invoked by *cron* to perform the disk accounting functions. By default, it will do disk accounting on the special files in **/etc/checklist.** If the **-o** flag is used, it does a slower version of disk accounting by login directory. *files* specifies the one or more filesystem names where disk accounting is to be done. If *files* is used, disk accounting will be done on these filesystems only. If the **-o** flag is used, *files* should be mount points of mounted filesystem. If omitted, they should be the special file names of mountable filesystems.

lastlogin is invoked by *runacct* (see *runacct*(1M)) to update **/usr/adm/acct/sum/loginlog**, which shows the last date on which each person logged in.

monacct should be invoked once each month or each accounting period. *number* indicates which month or period it is. If *number* is not given, it defaults to the current month (01 through12). This default is useful if *monacct* is to executed via *cron*(1M) on the first day of each month. *monacct* creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum**.

nulladm creates *file* with mode 664 and ensures that owner and group are **adm**. It is called by various accounting shell procedures.

prctmp can be used to print the session record file (normally **/usr/adm/acct/nite/ctmp** created by *acctcon1* (see *acctcon*(1M)).

prdaily is invoked by *runacct* (see *runacct*(1M)) to format a report of the previous day's accounting data. The report resides in **/usr/adm/acct/sum/rprt***mmdd* where *mmdd* is the month and day of the report. The current daily accounting reports may be printed by typing *prdaily*. Previous days' accounting reports can be printed by using the *mmdd* option and specifying the exact report date desired. The **-l** flag prints a report of exceptional usage by login id for the specifed date. Previous daily reports are cleaned up and therefore inaccessible after each invocation of *monacct*. The **-c** flag prints a report of exceptional resource usage by command, and can be used on current day's accounting data only.

prtacct can be used to format and print any total accounting (**tacct**) file.

shutacct should be invoked during a system shutdown (usually in **/etc/shutdown**) to turn process accounting off and append a "reason" record to **/etc/wtmp**.

startup should be called by **/etc/rc** to turn the accounting on whenever the system is brought up.

turnacct is an interface to *accton* (see *acct*(1M)) to turn process accounting **on** or **off**. The **switch** argument turns accounting off, moves the current **/usr/adm/pacct** to the next free name in **/usr/adm/pacct***incr* (where *incr* is a number starting with **1** and incrementing by one for each additional **pacct** file), then turns accounting back on again. This procedure is called by *ckpacct* and thus can be taken care of by the *cron* and used to keep **pacct** to a reasonable size.

FILES

/usr/lib/acct	holds all accounting commands listed in section (1M) of this manual
/usr/adm/fee	accumulator for fees
/usr/adm/acct/nite	working directory
/usr/adm/pacct	current file for per-process accounting
/usr/adm/pacct*	used if pacct gets large and during execution of daily accounting procedure
/usr/lib/acct/ptecms.awk	contains the limits for exceptional usage by command name
/usr/lib/acct/ptelus.awk	contains the limits for exceptional usage by login id
/usr/adm/acct/sum	summary directory, should be saved
/etc/wtmp	login/logoff summary

SEE ALSO

acct(1M), acctcms(1M), acctcom(1M), acctcon(1M), acctmerg(1M), acctprc(1M), cron(1M), diskusg(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

STANDARDS CONFORMANCE

chargefee: SVID2

ckpacct: SVID2

dodisk: SVID2

lastlogin: SVID2

monacct: SVID2

prctmp: SVID2

prdaily: SVID2

prtacct: SVID2

shutacct: SVID2

startup: SVID2

turnacct: SVID2 acctsh.1m

NAME

arp - address resolution display and control

SYNOPSIS

arp *hostname*
arp -a [*system*] [*core*]
arp -d *hostname*
arp -s *hostname address* [**temp**] [**pub**] [**trail**] [**rif** *rifAddress*]
arp -f *filename*

DESCRIPTION

arp displays and modifies the Internet-to-Ethernet address translation tables used by the Address Resolution Protocol.

OPTIONS

none — If no options are specified (first form above), **arp** displays the current ARP entry for *hostname*. The hostname must either appear in the hostname database (see *hosts*(4)), or be a DARPA Internet address expressed in Internet standard "dot notation".

-a — Display all current ARP entries by reading the table from file *core* (default **/dev/kmem**) based on the kernel file *system* (default **/hp-ux**).

-d — If an ARP entry exists for the host called *hostname*, delete it. This requires super-user privileges.

-s — Create an ARP entry for the host called *hostname* with the hardware station address *address*. The hardware station address is given as six hexadecimal bytes separated by colons. If an ARP entry already exists for *hostname*, the existing entry is updated with the new information. The entry is permanent unless the word **temp** is given in the command. If the word **pub** is specified, the entry is published, which means that this system will act as an ARP server responding to requests for *hostname* even though the host address is not its own. The word **trail** indicates that trailer encapsulations can be sent to this host. The word **rif** specifies source routing information used for token ring networks. This information allows a user to specify the paticular bridge route which the token ring packet should be delivered. *rifAddress* is given as an even number of hexadecimal bytes separated by colons, up to a maximum of 16 bytes. This requires super-user privileges.

-f — Read-file *filename* and set multiple entries in the ARP tables. Entries in the file should be of the form:

hostname address [**temp**] [**pub**] [**trail**] [**rif** *rifAddress*]

Argument meanings are the same as for the **-s** option.

AUTHOR

arp was developed by the University of California, Berkeley.

WARNINGS

HP 9000 systems can receive trailer packets but do not send them. Setting the trailers flag has no effect.

SEE ALSO

ifconfig(1M), inet(3N), lanconfig(1M), arp(7P).

NAME

Aserver - audio server

SYNOPSIS

`/usr/audio/bin/Aserver`

DESCRIPTION

Beginning with Release 8.07, the Series 700 HP-UX operating system includes audio software comprised of an Audio Application Program Interface (AAPI) and selected example programs.

The audio software package contains client and server components, which can run on separate systems. Audio data can reside on still a third system. In all cases, however, the server must run on a Series 700 system equipped with audio hardware. (To determine the presence of audio hardware, check the hardware manual provided with the system or look for audio jacks on the back of the computer enclosure.)

To set up the audio software correctly and ensure that the supporting services are running, follow these steps:

Step 1. Start the NCS Local Location Broker Daemon.

Task 1. If you are not already super-user, log in as super-user.

Task 2. Enter the command line `/usr/etc/ncs/llbd&`. To make `llbd` start automatically at boot time, edit the file `/etc/netncsrc` and change the line `START_LLBD=0` to `START_LLBD=1`.

Step 2. Reboot.

Step 3. Set the `AUDIO` environment variable to specify the node where the audio client should look for the audio server. If `AUDIO` is not set or if it is set to `:0`, the client connects with a server on the same node.

To specify that the client should connect to a server on another node, set

```
AUDIO = node_name:    (Korn, Bourne, and POSIX shells)
export AUDIO
```

or

```
setenv AUDIO node_name:    (C shell)
```

Step 4. Normally, the audio server starts whenever the system is booted. Check for the existence of the server processes by typing

```
ps -e | grep Aserver
```

You should see *two* `Aserver` processes. If the server is not running, start the audio server by hand by typing

```
/usr/audio/bin/Aserver
```

Then type

```
ps -e | grep Aserver
```

and check that there are two active server processes.

DEPENDENCIES

When an application program or the audio demonstration program uses the AAPI, the AAPI audio server component must run on a system that has audio hardware. Note that HP-UX for the 8-Mbyte Model 705 System does not include audio software.

AUTHOR

The AAPI and the audio demonstration program were developed by HP.

SEE ALSO

Audio(5).

Using the Audio Application Program Interface,
Audio Users Guide.

NAME

audevent - change or display event or system call audit status

SYNOPSIS

audevent [**-P** | **-p**] [**-F** | **-f**] [**-E**] [[**-e** *event*] ...] [**-S**] [[**-s** *syscall*] ...]

DESCRIPTION

audevent changes the auditing status of the given events or system calls. The *event* is used to specify names associated with certain self-auditing commands; *syscall* is used to select related system calls.

If neither **-P**, **-p**, **-F**, nor **-f** is specified, the current status of the selected events or system calls is displayed. If no events or system calls are specified, all events and system calls are selected.

If the **-E** option is supplied, it is redundant to specify events with the **-e** option; this applies similarly to the **-S** and **-s** options.

audevent takes effect immediately. However, the events and system calls specified are audited only when called by a user currently being audited (see *audusr*(1M)). A list of valid events and associated syscalls is provided in *audit*(5).

Only the super-user can change or display audit status.

Options

-P	Audit successful events or system calls.
-p	Do not audit successful events or system calls.
-F	Audit failed events or system calls.
-f	Do not audit failed events or system calls.
-E	Select all events for change or display.
-e *event*	Select *event* for change or display.
-S	Select all system calls for change or display.
-s *syscall*	Select *syscall* for change or display.

The following is a list of the valid *events* and the associated *syscalls* (if any):

create	Object creation (**creat, mkdir, mknod, msgget, pipe, semget, shmat, shmget**)
delete	Object deletion (**msgctl, rmdir, semctl**)
moddac	Discretionary access control (DAC) modification (**chmod, chown, fchmod, fchown, fsetacl, setacl, umask**)
modaccess	Non-DAC modification (**chdir, chroot, link, setgid, setuid, rename, setgroups, setresgid, setresuid, shmctl, shmdt, unlink**)
open	Object opening (**open, execv, execve, ptrace, truncate, ftruncate**)
close	Object closing (**close**)
process	Process operations (**fork, exit, kill, vfork, nsp_init**)
removable	Removable media events (**mount, umount, vfsmount**)
login	Logins and logouts
admin	administrative and superuser events (**audctl, audswitch, cluster, stime, reboot, setaudid, setaudproc, setdomainname, setevent, sethostid, setprivgrp, settimeofday, swapon**)
ipccreat	Interprocess Communication (IPC) object creation (**bind, ipccreate, ipcdest, socket**)
ipcopen	IPC object opening (**accept, connect, ipcconnect, ipclookup, ipcrecvn**)
ipcclose	IPC object deletion (**ipcshutdown, shutdown**)
ipcdgram	IPC datagram (**sendto, recvfrom**)

uevent1	User-defined event 1
uevent2	User-defined event 2
uevent3	User-defined event 3

AUTHOR

audevent was developed by HP.

SEE ALSO

audisp(1M), audomon(1M), audsys(1M), audusr(1M), getevent(2), setevent(2), audit(4), audit(5).

NAME

audisp - display the audit information as requested by the parameters

SYNOPSIS

audisp [**-u** *username*] [**-e** *eventname*] [**-c** *syscall*] [**-p**] [**-f**] [**-l** *ttyid*] [**-t** *start_time*] [**-s** *stop_time*] *audit_filename*(s) ...

DESCRIPTION

audisp analyzes and displays the audit information contained in the specified one or more audit files, *audit_filename*(s). The audit files are merged into a single audit trail in time order. Although the entire audit trail is analyzed, *audisp* allows you to limit the information displayed, by specifying options. This command is restricted to privileged users.

Each audit file specified can be a regular file or context-dependent file (CDF) consisting of several audit files from various cnodes (see *cdf*(4)). *audisp* recognizes a CDF and automatically merges all the files in the CDF into the audit trail.

Any unspecified option is interpreted as an unrestricted specification. For example, a missing **-u** *username* option causes all users' audit information in the audit trail to be displayed as long as it satisfies all other specified options. By the same principle, citing **-t** *start_time* without **-s** *stop_time* displays all audit information beginning from *start_time* to the end of the file.

audisp without any options displays all recorded information from the start of the audit file to the end.

Specifying an option without its required parameter results in error. For example, specifying **-e** without any *eventname* returns with an error message.

Options

-u *username* Specify the login name (*username*) about whom to display information. If no (*username*) is specified, *audisp* displays audit information about all users in the audit file.

-e *eventname* Display audit information of the specified event types. The defined event types are **create**, **delete**, **moddac**, **modaccess**, **open**, **close**, **process**, **removable**, **login**, **admin**, **ipccreat**, **ipcopen**, **ipcclose**, **uevent1**, **uevent2**, and **uevent3** (see *audevent*(1M)).

-c *syscall* Display audit information about the specified system calls.

-p Display only successful operations that were recorded in the audit trail. No user event that results in a failure is displayed, even if *username* and *eventname* are specified.

The **-p** and the **-f** options are mutually exclusive; do not specify both on the same command line. To display both successful and failed operations, omit both **-p** and **-f** options.

-f Display only failed operations that are recorded in the audit trail.

-l *ttyid* Display all operations that occurred on the specified terminal (*ttyid*) and were recorded in the audit trail. By default, operations on all terminals are displayed.

-t *start_time* Display all audited operations occurring since *start_time*, specified as *mmddhhmm*[*yy*] (month, day, hour, minute, year). If no year is given, the current year is used. No operation in the audit trail occurring before the specified time is displayed.

-s *stop_time* Display all audited operations occurring before *stop_time*, specified as *mmddhhmm*[*yy*] (month, day, hour, minute, year). If no year is given, the current year is used. No operation in the audit trail occurring after the specified time is displayed.

AUTHOR

audisp was developed by HP.

SEE ALSO

audevent(1M), audit(4), audit(5), cdf(4).

NAME

audomon - audit overflow monitor daemon

SYNOPSIS

/etc/audomon [**-p** *fss*] [**-t** *sp_freq*] [**-w** *warning*] [**-v**] [**-o** *output_tty*]

DESCRIPTION

audomon monitors the capacity of the current audit file and the file system on which the audit file is located, and prints out warning messages when either is approaching full. It also checks the audit file and the file system against 2 switch points: *FileSpaceSwitch* (FSS) and *AuditFileSwitch* (AFS) and if either is reached, audit recording automatically switches to the backup audit file if it is available.

The *FileSpaceSwitch* (FSS) is specified as a percentage of the total disk space available. When the file system reaches this percentage, *audomon* looks for a backup audit file. If it is available, recording is switched from the audit file to the backup file.

The *AuditFileSwitch* (AFS) is specified (using *audsys*(1M)) by the size of the audit file. When the audit file (or total CDF elements in a HP-UX clustered environment) reaches the specified size, *audomon* looks for a backup audit file. If it is available, recording is switched from the audit file to the backup file (see *audsys*(1M) for further information on use of this parameter).

If either switch point is reached but no backup file is available, *audomon* issues a warning message.

audomon is typically spawned by **/etc/auditrc** (as part of the *init*(1M) start-up process) when the system is booted up. Once invoked, *audomon* monitors, periodically sleeping and "waking up" at intervals. Note that *audomon* does not produce any messages when the audit system is disabled.

audomon is restricted to privileged users.

Options

-p *fss* Specify the *FileSpaceSwitch* by a number ranging from 0 to 100. When the audit file's file system has less than *fss* percent free space remaining, *audomon* looks for a backup file. If available, the backup file is designated as the new audit file. If no backup file is available, *audomon* issues a warning message.

The *fss* parameter should be a larger number than the *min_free* parameter of the file system to ensure that the switch takes place before *min_free* is reached. By default, *fss* is 20 percent.

-t *sp_freq* Specify the wake-up switch-point frequency in minutes. The wake-up frequency at any other time is calculated based on *sp_freq* and the current capacity of the audit file and the file system. The calculated wake-up frequency at any time before the switch points is larger than *sp_freq*. As the size of the audit file or the file system's free space approaches the switch points, the wake-up frequency approaches *sp_freq*. *sp_freq* can be any positive real number. Default *sp_freq* is 1 (minute).

-w *warning* Specify that warning messages be sent before the switch points. *warning* is an integer ranging from 0 through 100. The higher the *warning*, the closer to the switch points warning messages are issued. For example, *warning* = 50 causes warning messages to be sent half-way before the switch points are reached. *warning* = 100 causes warning messages to be sent only after the designated switch points are reached and a switch is not possible due to a missing backup file. By default, *warning* is 90.

-v Make audomon more verbose. This option causes *audomon* to also print out the next wake-up time.

-o *output_tty* Specify the tty to which warning messages are directed. By default, warning messages are sent to the console. Note that this applies only to the diagnostic messages *audomon* generates concerning the status of the audit system. Error messages caused by wrong usage of *audomon* are sent to the standard output (where *audomon* is invoked).

AUTHOR

audomon was developed by HP.

SEE ALSO

audsys(1M), audit(5).

NAME

audsys - start or halt the auditing system and set or display audit file information

SYNOPSIS

audsys [**-nf**] [**-c** *file* **-s** *cafs*] [**-x** *file* **-z** *xafs*]

DESCRIPTION

audsys allows the user to start or halt the auditing system, to specify the auditing system "current" and "next" audit files (and their switch sizes), or to display auditing system status information. This command is restricted to super-users.

The "current" audit file is the file to which the auditing system writes audit records. When the "current" file grows to either its Audit File Switch (AFS) size or its File Space Switch (FSS) size (see *audomon*(1M)), the auditing system switches to write to the "next" audit file. The auditing system switches audit files by setting the "current" file designation to the "next" file and setting the new "next" file to NULL. The "current" and "next" files can reside on different file systems.

When invoked without arguments, *audsys* displays the status of the auditing system. This status includes information describing whether auditing is on or off, the names of the "current" and "next" audit files, and a table listing their switch sizes and the sizes of file systems on which they are located, as well as the space available expressed as a percentage of the switch sizes and file system sizes.

Options

audsys recognizes the following options:

-n Turn on the auditing system. The system uses existing "current" and "next" audit files unless others are specified with the **-c** and **-x** options. If no "current" audit file exists (such as when the auditing system is first installed), specify it by using the **-c** option.

-f Turn off the auditing system. The **-f** and **-n** options are mutually exclusive. Other options specified with **-f** are ignored.

-c *file* Specify a "current" file. Any existing "current" file is replaced with the *file* specified; the auditing system immediately switches to write to the new "current" file. The specified *file* must be empty or nonexistent, unless it is the "current" or "next" file already in use by the auditing system.

-s *cafs* Specify *cafs*, the "current" audit file switch size (in kbytes).

-x *file* Specify the "next" audit file. Any existing "next" file is replaced with the *file* specified. The specified *file* must be empty or nonexistent, unless it is the "current" or "next" file already in use by the auditing system.

-z *xafs* Specify *xafs*, the "next" audit file switch size (in kbytes).

If **-c** but not **-x** is specified, only the "current" audit file is changed; the existing "next" audit file remains. If **-x** but not **-c** is specified, only the "next" audit file is changed; the existing "current" audit file remains.

The **-c** option can be used to manually switch from the "current" to the "next" file by specifying the "next" file as the new "current" file. In this instance, the file specified becomes the new "current" file and the "next" file is set to NULL.

In instances where no next file is desired, the **-x** option can be used to set the "next" file to NULL by specifying the existing "current" file as the new "next" file.

The user should take care to select audit files that reside on file systems large enough to accomodate the Audit File Switch (AFS) desired. *audsys* returns a non-zero status and no action is performed, if any of the following situations would occur:

The Audit File Switch size (AFS) specified for either audit file exceeds the space available on the file system where the file resides.

The AFS size specified for either audit file is less than the file's current size.

Either audit file resides on a file system with no remaining user space (exceeds minfree).

AUTHOR

audsys was developed by HP.

FILES

/.secure/etc/audnames File maintained by *audsys* containing the "current" and "next" audit file names and their switch sizes.

SEE ALSO

audit(5), audomon(1M), audctl(2), audwrite(2), audit(4).

NAME

audusr - select users to audit

SYNOPSIS

audusr [[**-a** *user*] ...][[**-d** *user*] ...][**-A**|**-D**]

DESCRIPTION

audusr is used to specify *user*s to be audited or excluded from auditing. If no arguments are specified, **audusr** displays the command usage. **audusr** is restricted to super-users.

Options

audusr recognizes the following options:

-a *user*	Audit the specified *user*. The auditing system records audit records to the "current" audit file when the specified *user* executes audited events or system calls. Use **audevent** to specify events to be audited (see *audevent*(1M)).
-d *user*	Do not audit the specified *user*.
-A	Audit all users.
-D	Do not audit any users.

The **-A** and **-D** options are mutually exclusive: that is, if **-A** is specified, **-d** cannot be specified; if **-D** is specified, **-a** cannot be specified.

Users specified with **audusr** are audited (or excluded from auditing) beginning with their next login session, until excluded from auditing (or specified for auditing) with a subsequent **audusr** invocation. Users already logged into the system when **audusr** is invoked are unaffected during that login session; however, any user who logs in after **audusr** is invoked is audited or excluded from auditing accordingly.

AUTHOR

audusr was developed by HP.

FILES

/.secure/etc/passwd
File containing flags to indicate whether users are audited.

SEE ALSO

audit(5), audevent(1M), setaudproc(2), audswitch(2), audwrite(2).

NAME

automount - automatically mount NFS file systems

SYNOPSIS

automount [**-mnTv**] [**-D** *name = value*] [**-f** *master-file*] [**-M** *mount-directory*] [**-tl** *duration*] [**-tm** *interval*] [**-tw** *interval*] [*directory map* [*-mount-options*]] ...

DESCRIPTION

automount is a daemon that automatically and transparently mounts NFS file systems as needed. It monitors attempts to access directories that are associated with an **automount** map, along with any directories or files that reside under them. When a file is to be accessed, the daemon mounts the appropriate NFS file system. Maps can be assigned to a directory by using an entry in a direct **automount** map, or by specifying an indirect map on the command line.

automount interacts with the kernel in a manner closely resembling an NFS server:

- **automount** uses the map to locate an appropriate NFS file server, exported file system, and mount options.
- It then mounts the file system in a temporary location, and replaces the file system entry for the directory or subdirectory with a symbolic link to the temporary location.
- If the file system is not accessed within an appropriate interval (five minutes by default), the daemon unmounts the file system and removes the symbolic link.
- If the specified directory has not already been created, the daemon creates it, and then removes it upon exiting.

Since name-to-location binding is dynamic, updates to an **automount** map are transparent to the user. This obviates the need to mount shared file systems prior to running applications that contain internally hard-coded references to files.

If the dummy directory (**/-**) is specified, **automount** treats the *map* argument that follows as the name of a direct map. In a direct map, each entry associates the full pathname of a mount point with a remote file system to mount.

If the *directory* argument is a pathname, the *map* argument points to an indirect map. An indirect map, contains a list of the subdirectories contained within the indicated *directory*. With an indirect map, it is these subdirectories that are mounted automatically.

A map can be a file or a NIS map; if a file, the *map* argument must be a full pathname.

The *-mount-options* argument, when supplied, is a comma-separated list of options to the **mount** command (see *mount*(1M)) preceded by a **-**. However, any conflicting mount options specified in the indicated map take precedence.

Options

automount recognizes the following options:

-m Suppress initialization of *directory-map* pairs listed in the **auto.master** NIS database.

-n Disable dynamic mounts. With this option, references through the **automount** daemon succeed only when the target filesystem has been previously mounted. This can be used to prevent NFS servers from cross-mounting each other.

-T Trace. Expand each NFS call and display it on the standard error.

-v Verbose. Log status messages to the system log file (see *syslogd*(1M)).

-D *envar = value*
Assign *value* to the indicated **automount** (environment) variable *envar*.

-f *master-file* Read a local file for initialization, ahead of the **auto.master** NIS map.

-M *mount-directory*
Mount temporary file systems in the named directory instead of in **/tmp_mnt**.

-tl *duration* Specify a *duration* (in seconds) that a file system is to remain mounted when not in use. The default is 5 minutes.

`-tm` *interval* Specify an *interval* (in seconds) between attempts to mount a filesystem. The default is 30 seconds.

`-tw` *interval* Specify an *interval* (in seconds) between attempts to unmount filesystems that have exceeded their cached times. The default is 1 minute.

ENVIRONMENT

Environment variables can be used within an `automount` map. For example, if `$HOME` appears within a map, `automount` expands it to the current value of the `HOME` environment variable.

To protect a reference from affixed characters, surround the variable name with curly braces. Environment variables cannot appear as the key entry in maps.

USAGE

Map Entry Format

A simple map entry (mapping) takes the form:

directory [*-mount-options*] *location* ...

where *directory* is the full pathname of the directory to mount, when used in a direct map, or the basename of a subdirectory in an indirect map. *mount-options* is a comma-separated list of `mount` options, and *location* specifies a remote filesystem from which the directory may be mounted. In the simple case, *location* takes the form:

host : *pathname*

Multiple *location* fields can be specified, in which case `automount` sends multiple `mount` requests; `automount` mounts the file system from the first host that replies to the `mount` request. This request is first made to the local net or subnet. If there is no response, any connected server is allowed to respond.

If *location* is specified in the form:

host : *path* : *subdir*

host is the name of the host from which to mount the file system, *path* is the pathname of the directory to mount, and *subdir*, when supplied, is the name of a subdirectory to which the symbolic link is made. This can be used to prevent duplicate mounts when multiple directories in the same remote file system might be accessed. Assume a map for `/users` resembling:

```
mike        hpserver1:/users/hpserver1:mike
dianna      hpserver1:/users/hpserver1:dianna
```

Attempting to access a file in `/users/mike` causes `automount` to mount `hpserver1:/users/hpserver1` and creates a symbolic link called `/users/mike` to the `mike` subdirectory in the temporarily-mounted filesystem. A subsequent file access request in `/users/dianna` results in `automount` simply creating a symbolic link that points to the `dianna` subdirectory because `/users/hpserver1` is already mounted. Given the map:

```
mike        hpserver1:/users/hpserver1/mike
dianna      hpserver1:/users/hpserver1/dianna
```

`automount` would have to mount the filesystem twice.

A mapping can be continued across input lines by escaping the new-line character with a backslash (\). Comments begin with a `#` and end at the subsequent new-line character.

Directory Pattern Matching

The `&` character is expanded to the value of the *directory* field for the entry in which it occurs. Given an entry of the form:

```
mike        hpserver1:/users/hpserver1:&
```

the `&` expands to `mike`.

The `*` character, when supplied as the *directory* field, is recognized as the catch-all entry. Such an entry resolves to any entry not previously matched. For example, if the following entry appeared in the indirect map for `/users`:

```
*    &:/users/&
```

this would allow automatic mounts in **/users** of any remote file system whose location could be specified as:

hostname **:/users/***hostname*

Hierarchical Mappings

A hierarchical mapping takes the form:

directory [*/*[*subdirectory*] [*-mount-options*] *location* ...] ...

The initial **/** within the */*[*subdirectory*] is required; the optional *subdirectory* is taken as a filename relative to the *directory*. If *subdirectory* is omitted in the first occurrence, the **/** refers to the directory itself.

Given the direct map entry:

```
/usr/local    \
    /      -ro,intr    shasta:/usr/local        ranier:/usr/local      \
    /bin   -ro,intr    ranier:/usr/local/bin    shasta:/usr/local/bin \
    /man   -ro,intr    shasta:/usr/local/man    ranier:/usr/local/man
```

automount automatically mounts **/usr/local**, **/usr/local/bin**, and **/usr/local/man**, as needed, from either **shasta** or **ranier**, whichever host responded first.

Direct Maps

A direct map contains mappings for any number of directories. Each directory listed in the map is automatically mounted as needed. The direct map as a whole is not associated with any single directory.

Indirect Maps

An indirect map allows specifying mappings for the subdirectories to be mounted under the *directory* indicated on the command line. It also obscures local subdirectories for which no mapping is specified. In an indirect map, each *directory* field consists of the basename of a subdirectory to be mounted as needed.

Included Maps

The contents of another map can be included within a map with an entry of the form:

+mapname

mapname can either be a filename, or the name of an NIS map, or one of the special maps described below.

Special Maps

Three special maps, **-hosts**, **-passwd**, and **-null**, are currently available: The **-hosts** map uses the **gethostbyname()** map to locate a remote host when the hostname is specified (see *gethostbyname*(3C). This map specifies mounts of all exported file systems from any host. For example, if the following **automount** command is already in effect:

```
automount /net -hosts
```

a reference to **/net/hermes/usr** initiates an automatic mount of all file systems from **hermes** that **automount** can mount, and any subsequent references to a directory under **/net/hermes** refer to the corresponding directory on **hermes**. The **-passwd** map uses the *passwd*(4) database to attempt to locate a user's home directory. For example, if the following **automount** command is already in effect:

```
automount /homes -passwd
```

if the home directory for a user has the form */dir/server/username*, and *server* matches the host system on which that directory resides, **automount** mounts the user's home directory as: **/homes/***username*.

For this map, the tilde character (~) is recognized as a synonym for *username*.

The **-null** map, when indicated on the command line, cancels a previous map for the directory indicated. It can be used to cancel a map given in **auto.master**.

Configuration and the auto.master Map

automount normally consults the **auto.master** NIS configuration map for a list of initial **automount** maps, and sets up automatic mounts for them in addition to those given on the command line. If there are duplications, the command-line arguments take precedence. This configuration database contains arguments to the **automount** command rather than mappings. Unless **-f** is in effect, **automount** does *not* look for an **auto.master** file on the local host.

Maps given on the command line, or those given in a local **auto.master** file specified with **-f** override those in the NIS **auto.master** map. For example, given the command:

```
automount /homes /etc/auto.homes /- /etc/auto.direct
```

and the NIS map file **auto.master** containing:

```
/homes -passwd
```

automount mounts home directories using the **/etc/auto.homes** map instead of the special **-passwd** map in addition to the various directories specified in the **/etc/auto.direct** map.

WARNINGS

Do not send the **SIGKILL** signal (**kill -9**, or **kill -KILL**) to the automount daemon. Doing so causes any processes accessing mount directories served by **automount** to hang. A system reboot may be required to recover from this state.

Do not start an automount daemon while another is still running. If restarting **automount**, make sure the first daemon and all of its children are not running.

When **automount** receives signal **SIGHUP**, it rereads the **/etc/mnttab** file to update its internal record of currently mounted file systems. If a file system mounted by **automount** is unmounted by a **umount** command, **automount** should be forced to reread the file by sending the **SIGHUP** signal (see **kill** (1)).

Shell filename expansion does not apply to objects not currently mounted.

Since **automount** is single-threaded, any request that is delayed by a slow or non-responding NFS server delays all subsequent automatic mount requests until it completes.

Programs that read **/etc/mnttab** and then touch files that reside under automatic mount points introduce further entries to the file.

Automatically-mounted file systems are mounted with type **ignore**; they do not appear in the output of either *mount*(1M), or *bdf*(1M).

FILES

/tmp_mnt	directory under which filesystems are dynamically mounted
/etc/mnttab	mount table

SEE ALSO

mount(1M), **bdf**(1M), **passwd**(4)

NAME

backup - backup or archive file system

SYNOPSIS

/etc/backup [**-A**] [*-archive*] [**-fsck**]

DESCRIPTION

backup uses *find*(1) and *cpio*(1) to save a *cpio* archive of all files that have been modified since the modification time of **/etc/archivedate** on the default tape drive (**/dev/rct**). *backup* should be invoked periodically to ensure adequate file backup.

The **-A** option suppresses warning messages regarding optional access control list entries. *backup*(1M) does not backup optional access control list entries in a file's access control list (see *acl*(5)). Normally, a warning message is printed for each file having optional access control list entries.

The **-archive** option causes *backup* to save all files, regardless of their modification date, and then update **/etc/archivedate** using *touch*(1).

backup prompts you to mount a new tape and continue if there is no more room on the current tape. Note that this prompting does not occur if you are running *backup* from *cron*(1M).

The **-fsck** option causes *backup* to start a file system consistency check (without correction) after the backup is complete. For correct results, it is important that the system be effectively single-user while *fsck* is running, especially if **-fsck** is allowed to automatically fix whatever inconsistencies it finds. *backup* does not ensure that the system is single-user.

You can edit **/etc/backup** to customize it for your system. For example, *backup* uses *tcio*(1) with *cpio* to back up files on an HP CS/80 disc drive's streaming tape. You must modify *backup* to use *cpio*(1) if you want to access a standard HP Tape Drive.

Several local values are used that can be customized:

backupdirs	specifies which directories to back up recursively (usually /, meaning all directories);
backuplog	file name where start and finish times, block counts, and error messages are logged;
archive	file name whose date is the date of the last archive;
remind	file name that is checked by **/etc/profile** to remind the next person who logs in to change the backup tape;
outdev	specifies the output device for the backed-up files;
fscklog	file name where start and finish times and *fsck* output is logged.

You may want to make other changes, such as whether or not *fsck* does automatic correction (according to its arguments), where *cpio* output is directed, other information logging, etc.

In all cases, the output from *backup* is a normal *cpio* archive file (or volume) which can be read using *tcio* and *cpio* with the **c** option.

File Recovery

backup creates archive tapes with all files and directories specified relative to the root directory. When recovering files from an archive tape created by *backup*, you should be in the root directory and specify the directory path names for recovered files relative to the root directory (/). When specifying the directory path name for file recovery by *tcio* and *cpio*, do not precede the leading directory name with a slash. If you prefer, you can also use *cpio* with a **-t** option to determine how files and directories are named on the archive tape before attempting recovery.

WARNINGS

Refer to WARNINGS in *cpio*(1).

When *cpio* runs out of tape, it sends an error to standard error and demands a new special file name from **/dev/tty**.

To continue, rewind the tape, mount the new tape, type the name of the new special file at the system console, and press **Return**.

If *backup* is being run unattended from *cron*(1M) and the tape runs out, *backup* terminates, leaving the *find* process still waiting. Kill this process when you return.

FILES

/etc/archivedate parameterized file names

SEE ALSO

cpio(1), find(1), tcio(1), touch(1), cron(1M), fbackup(1M), frecover(1M), fsck(1M), acl(5).

NAME

bdf - report number of free disk blocks (Berkeley version)

SYNOPSIS

bdf [**-b**][**-i**][**-l**|**-L**][**-t** *type* | [*filesystem*|*file*] ...]

DESCRIPTION

bdf prints out the amount of free disk space available on the specified *filesystem* (**/dev/dsk/c0d0s0**, for example) or on the file system in which the specified *file* (**$HOME**, for example) is contained. If no file system is specified, the free space on all of the normally mounted file systems is printed. The reported numbers are in kilobytes.

Options

bdf recognizes the following options:

-b Display information regarding file system swapping.

-i Report the number of used and free inodes.

-l In the HP Clustered environment, display information for only HFS and CDFS file systems mounted on the local cnode (NFS mounts are not displayed).

-L In the HP Clustered environment, display information for file systems that can be unmounted from the local cnode (includes file systems mounted on the local node and cluster-wide NFS mounts).

-t *type* Report on the filesystems of a given *type* (for example, **nfs** or **hfs**).

WARNINGS

If file system names are too long, the output for a given entry is displayed on two lines.

bdf does not account for any disk space reserved for swap space, or used for the HFS boot block (8k bytes, 1 per file system), HFS superblocks (8k bytes each, 1 per disk cylinder), HFS cylinder group blocks (1k-8k bytes each, 1 per cylinder group), and inodes (currently 128 bytes reserved for each inode). Non-HFS file systems may have other items that this command does not account for.

In the HP Clustered environment, device and mountpoint pathnames are displayed with CDFs expanded.

AUTHOR

bdf was developed by the University of California, Berkeley.

FILES

/etc/checklist	static information about the file systems
/etc/mnttab	mounted file system table
/dev/dsk/*	file system devices

SEE ALSO

df(1M), checklist(4), mnttab(4).

NAME

bifdf - report number of free disk blocks

SYNOPSIS

bifdf [**-t**][**-f**][*file-systems*]

DESCRIPTION

bifdf prints out the number of free blocks and free inodes available for on-line Bell file systems by examining the counts kept in the super-blocks. *file-systems* can be specified by device name; e.g., **/dev/dsk/1s0**.

Options

bifdf recognizes the following options:

-t Report the total allocated block figures as well.

-f Make only an actual count of the blocks in the free list (free inodes are not reported). With this option, **bifdf** reports on raw devices.

AUTHOR

bifdf was developed by HP.

SEE ALSO

biffsck(1M), df(1M), bif(4).

NAME

biffsck - Bell file system consistency check and interactive repair

SYNOPSIS

biffsck [**-y**] [**-n**] [**-s***nb_c*:*nb_s*] [**-S***nb_c*:*nb_s*] [**-t***filename*] [*file-system*] ...

DESCRIPTION

biffsck audits and interactively repairs inconsistent conditions in a Bell file system. If the file system is consistent, the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent, the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions 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 the operator to respond **yes** or **no**. If the operator does not have write permission, *biffsck* defaults to the **-n** option described below.

The following flags are interpreted by *biffsck*.

-y Assume a **yes** response to all questions asked by *biffsck*.

-n Assume a **no** response to all questions asked by *biffsck*, and do not open the file system for writing.

-s*nb_c*:*nb_s*

Ignore the actual free list and unconditionally reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done.

This option allows for creating an optimal free-list organization. Arguments to the -s option are numerical values separated by a colon and are determined as follows:

nb_c Number of blocks per cylinder.

nb_s Number of blocks to skip.

If *nb_c*:*nb_s* is not specified with this option, the values used when the file system was created are used. If no values were specified when the file system was created, the following defaults are used for the devices indicated:

Model	*Blocks/cyl*:*Blocks to skip*
HP 7908A	35:2
HP 7933A	23:15
HP 7911A	16:12
HP 7912A	16:12
HP 7914A	16:12
Default for *biffsck*(1M)	400:9
Default for *bifmkfs*(1M)	500:3

-S*nb_c*:*nb_s*

Conditionally reconstruct the free list. This option is similar to **-s***nb_c*:*nb_s* above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **-S** forces a **no** response to all questions asked by *biffsck*. This option is useful for forcing free list reorganization on uncontaminated Bell file systems.

-t If *biffsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **-t** option is specified, the file named in the accompanying *filename* argument is used as the scratch file, if needed. Without the **-t** flag, *biffsck* prompts the operator for the name of the scratch file. The file chosen should not be on the file system being checked. If the file does not exist, *biffsck* creates it. If the scratch file is not a special file, it is removed when *biffsck* terminates.

file-system is a device file name on which the file system to be checked resides; for example, **/dev/dsk/1s0**.

biffsck checks for the following inconsistencies:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.

4. Size checks:
   Incorrect number of blocks.
   Directory size not 16-byte aligned.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated inode.
   Inode number out of range.
8. Super Block checks:
   More than 65536 inodes.
   More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **/lost+found** directory on the BIF volume. The name assigned is the inode number. The only restriction is that the directory **lost+found** must already exist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory (optimally in multiples of 64), then removing them before *biffsck* is executed.

biffsck can check file systems on both raw and blocked devices. Checking raw devices is almost always faster, but should not be used on a mounted file system.

DIAGNOSITCS

The diagnostics produced by *biffsck* are intended to be self-explanatory.

WARNINGS

Inode numbers for . and .. in each directory should be checked for validity.

AUTHOR

biffsck was developed by HP.

SEE ALSO

bif(4).

NAME

biffsdb - Bell file system debugger

SYNOPSIS

biffsdb *special* [-]

DESCRIPTION

biffsdb can be used to patch a damaged Bell file system after a crash or failure. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an inode. These greatly simplify the process of correcting control block entries or descending the Bell file system tree.

biffsdb contains several error checking routines to verify inode and block addresses. These can be disabled if necessary by invoking *biffsdb* with the optional - argument or by the use of the **O** symbol (*biffsdb* reads the i-size and f-size entries from the super-block of the file system as the basis for these checks).

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

biffsdb reads a block at a time and therefore works with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

biffsdb recognizes the following symbols:

#	absolute address
i	convert from i-number to inode address
b	convert to block address
d	directory slot offset
+,-	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=-	decremental assignment
="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The following print options are available:

i	print as inodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes

The **f** symbol is used to print data blocks associated with the current inode. If followed by a number, that block of the file is printed (blocks are numbered from zero). The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs and spaces can be used as function delimiters but are not necessary. A line with just a new-line character increments the current address by the size of the data type last printed; that is, the address is set to the next byte, word, double word, directory entry or inode, enabling the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. Inodes are printed with labeled fields describing each element.

The following mnemonics are used for inode examination and refer to the current working inode:

md	mode
ln	link count
uid	user ID number
gid	group ID number
s0	high byte of file size
s1	low word of file size
a#	data block numbers (0 - 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i	Print i-number 386 in an inode format. This now becomes the current working inode.
ln=4	Change the link count for the working inode to 4.
ln=+1	Increment the link count by 1.
fc	Print, in ASCII, block zero of the file associated with the working inode.
2i.fd	Print the first 32 directory entries for the root inode of this file system.
d5i.fc	Change the current inode to that associated with the 5th directory entry (numbered from zero) found from the above command. The first 512 bytes of the file are then printed in ASCII.
1b.p0o	Print the superblock of this file system in octal.
2i.a0b.d7=3	Change the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
d7.nm="*name***"**	Change the *name* field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.

AUTHOR

biffsdb was developed by HP.

SEE ALSO

bif(4), biffsck(1M).

NAME

bifmkfs - construct a Bell file system

SYNOPSIS

bifmkfs *special blocks* [:*inodes*] [*gap blocks*]
bifmkfs *special proto* [:*inodes*] [*gap blocks*]

DESCRIPTION

bifmkfs constructs a Bell file system by writing on the special file according to the directions found in the remainder of the command line. If the second argument is given as a string of digits, *bifmkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *blocks* interpreted as a decimal number. The boot program is left uninitialized. If the optional number of inodes is not given, the default is the number of blocks divided by 4.

If the second argument is a file name that can be opened, *bifmkfs* assumes it to be a prototype file *proto*, and will take its directions from that file. The prototype file contains tokens separated by spaces or newlines. The first token is the name of a file to be copied onto block zero as the bootstrap program. The second token is a number specifying the size of the created file system. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the i-list size in blocks The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **-bcd** specify regular, block special, character special and directory files respectively.) The second character of the type 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 three digit octal number giving the owner, group, and other read, write, execute permissions, see *bifchmod*(1).

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 is a path name that designates the file from which the contents and size are copied. If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers. If the file is a directory, *bifmkfs* makes the entries . and .. and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
/stand/diskboot
4872 110
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

In both command syntaxes, the rotational *gap* and the number of *blocks* can be specified. (For RP04 type drives, these numbers should be 7 and 418.)

EXAMPLES

To put a Bell file system on a disk with 770 1K blocks of capacity:

bifmkfs /dev/rdsk/1s0 770

where **/dev/rdsk/1s0** is the device special file for the micro floppy.

WARNINGS

If a prototype is used, it is not possible to initialize a file with second- or third-level indirects.

AUTHOR
bifmkfs was developed by HP.

SEE ALSO
bif(4).

NAME

boot - bootstrap process

DESCRIPTION

The Series 700 and 800 bootstrap process involves the execution of three software components:

- **pdc** (see *pdc*(1M),
- **isl** (see *isl*(1M), and
- **hpux** (see *hpux_800*(1M)).

After the processor is RESET, **pdc**, the **processor-dependent code** (firmware), performs a self-test and initializes the processor. It then loads and transfers control to **isl**, the operating-system-independent *initial system loader*. **isl**, in turn, loads and transfers control to the **hpux** utility, the HP-UX-specific bootstrap loader. **hpux** then downloads the HP-UX kernel object file from an HP-UX file system and transfers control to the loaded kernel image.

SEE ALSO

hpux(1M), hpux_800(1M), isl(1M), pdc(1M).

(Requires Optional ARPA Services Software)

NAME

bootpd - Internet Boot Protocol server

SYNOPSIS

/etc/bootpd [**-t** *timeout*] [**-s**] [**-d** *debuglevel*] [*configfile* [*dumpfile*]]

DESCRIPTION

bootpd implements an Internet Boot Protocol (BOOTP) server as defined in RFC951 and RFC1048. *bootpd* can be run either through *inetd*(1M) or as a stand-alone daemon. It is run by **/etc/inetd** when the following line in included the file **/etc/inetd.conf**:

bootps dgram udp wait root /etc/bootpd bootpd

bootpd starts when a boot request arrives. If *bootpd* does not receive another boot request within fifteen minutes of the last one received, it exits. The **-t** option can be used to specify a different timeout value in minutes (such as **-t20**). A timeout value of zero means that *bootpd* will never exit.

To run *bootpd* as a stand-alone daemon, invoke it with the **-s** option (for example, at boot time from **/etc/netbsdsrc**). This may be the desired mode of operation for large network installations with many BOOTP clients. In this case, the **-t** option has no effect since *bootpd* never exits.

The **-d** option sets the verbosity level of the logging emitted by the daemon.

Configuration

Upon startup, *bootpd* reads its configuration file, **/etc/bootptab**, or the *configfile* specified in the command line, then listens for boot request packets. *bootpd* rereads its configuration file when it receives a hangup signal, SIGHUP, or when it receives a boot request packet and detects that the file has been updated. If hosts are added, deleted or modified, their entries in *bootpd*'s internal database are updated accordingly when the configuration file is reread.

If *bootpd* receives a SIGUSR1 signal, it dumps its memory-resident database to the file **/etc/bootpd.dump** or the *dumpfile* specified in the command line.

The configuration file uses two-character, case-sensitive tag symbols to represent host parameters. These parameter declarations are separated by colons (**:**). The general format is:

hostname:*tg*=*value*:...:*tg*=*value*:...:*tg*=*value*:...

where *hostname* is the actual name of a BOOTP client and *tg* is a two-character tag symbol. Most tags must be followed by an equals-sign and a value as above. Some can also appear in a boolean form with no value (i.e. :*tg*:).

Blank lines and lines beginning with "#" are ignored in the configuration file. Host entries are separated from one another by newlines; a single host entry can be extended over multiple lines if the lines end with a backslash (\). It is also acceptable for lines to be longer than 80 characters. Tags can appear in any order with the following exceptions: the hostname must be the very first field in an entry, nd the hardware type tag, **ht**, must precede the hardware address tag, **ha**.

IP addresses are specified in standard Internet "dot" notation, and can use decimal, octal, or hexadecimal numbers (octal numbers begin with **0**, hexadecimal numbers begin with **0x** or **0X**). Certain tags accept a list of one or more IP addresses (*ip-address-list*). When more than one IP address is listed, the addresses should be separated by white space.

The currently recognized tags are:

ba This tag specifies that *bootpd* should broadcast the boot reply to the client. As a boolean tag, it causes *bootpd* to send the boot reply on the configured broadcast address of each network interface. You can also assign the tag an ip-address value which specifies the specific IP or broadcast address for the boot reply. The **ba** tag should only be used for diagnostic purposes; e.g., for debugging boot replies with *bootpquery*(1M).

bf=*filename*

This tag specifies the *filename* of the bootfile which the client should download via TFTP. The client's boot request and the values of the **hd** (see below) and **bf** symbols determine how the server fills in the bootfile field of the boot reply packet.

If the client specifies an absolute pathname and that file exists on the server machine, that pathname is returned in the reply packet. If the file cannot be found, the request is discarded;

no reply is sent. If the client specifies a relative pathname, a full pathname is formed by appending the pathname to the value of the **hd** tag and testing for existence of the file. If the resulting boot file pathname cannot be found, the request is discarded.

Clients that do not specify boot files in their boot requests will always elicit a reply from the server. The exact reply will again depend upon the **hd** and **bf** tags. If the **bf** tag gives an absolute pathname and the file exists, that pathname is returned in the reply packet. Otherwise, if the **hd** and **bf** tags together specify an accessible file, that filename is returned in the reply. If a complete filename cannot be determined or the file does not exist, the reply will contain a zeroed-out bootfile field.

tftpd(1m) performs a *chroot*(2) to the home directory of the pseudo-user *tftp*. when this pseudo-user exists, *bootpd* treats all absolute pathnames as being relative to this user's home directory. In all cases, existence of a file means that, in addition to actually being present, the file must be readable by the pseudo-user *tftp* (or must be publicly readable, if no such pseudo-user exists) since this is also required by *tftpd*(1M) to permit the file transfer. All filenames are first tried as *filename.hostname* and then simply as *filename*, thus providing for individual per-host bootfiles.

bs=*size*
This tag specifies the size of the bootfile. The parameter *size* can be either a decimal, octal, or hexadecimal integer specifying the size of the bootfile in 512-octet blocks, or the keyword **auto** which causes the server to automatically calculate the bootfile size at each request. Specifying the **bs** symbol as a boolean has the same effect as specifying **auto** as its value.

cs=*ip-address-list*
This tag specifies the IP address(es) of RFC865 Quote of the Day (cookie) server(s).

ds=*ip-address-list*
This tag specifies the IP address(es) of RFC1034 Domain Name server(s).

gw=*ip-address-list*
This tag specifies the IP address(es) of gateway(s) for the client's subnet. If one of many gateways is preferred, it should be listed first.

ha=*hardware-address*
This tag specifies the hardware address of the client. The *hardware address* must be specified in hexadecimal; optional periods and/or a leading **0x** can be included for readability. The **ha** tag must be preceded by the **ht** tag (either explicitly or implicitly; see **tc** below).

hd=*home-directory*
This tag specifies a directory name to which the bootfile is appended (see **bf** tag above). The default value of the **hd** tag is /.

hn The presence of this tag indicates that the hostname should be sent in the boot reply. The **hn** tag is a boolean tag. *bootpd* attempts to send the entire hostname as it is specified in the configuration file; if this cannot fit into the reply packet, an attempt is made to shorten the name to just the host field (up to the first period, if present) and then tried. In no case is an arbitrarily-truncated hostname sent. If nothing reasonable can fit, nothing is sent.

ht=*hardware-type*
This tag specifies the hardware type code. The *hardware-type* parameter can be an unsigned decimal, octal, or hexadecimal integer corresponding to one of the ARP Hardware Type codes specified in RFC1010. It can also be specified by the symbolic names **ethernet** or **ether** for 10-Mb Ethernet; **ethernet3** or **ether3** for 3-Mb experimental Ethernet; **ieee802**, **tr**, or **token-ring** for IEEE 802 networks; **pronet** for Proteon ProNET Token Ring; **chaos**, **arcnet**, or **ax.25** for Chaos, ARCNET, and AX.25 Amateur Radio networks, respectively.

im=*ip-address-list*
This tag specifies the IP address(es) of Impress network image server(s).

ip=*ip-address*
This tag specifies the IP address of the BOOTP client.

lg=*ip-address-list*
This tag specifies the IP address(es) of MIT-LCS UDP log server(s).

(Requires Optional ARPA Services Software)

lp=*ip-address-list*
This tag specifies the IP address(es) of Berkeley 4BSD printer server(s).

ns=*ip-address-list*
This tag specifies the IP address(es) of IEN-116 name server(s).

rl=*ip-address-list*
This tag specifies the IP address(es) of RFC887 Resource Location Protocol server(s).

sm=*subnet-mask*
This tag specifies the client's subnet mask. The *subnet-mask* is specified as a single IP address.

T*nnn*=*generic-data*
This is a generic tag where *nnn* is an RFC1048 vendor field tag number. This allows *bootpd* to immediately take advantage of future extensions to RFC1048. The *generic-data* data can be represented as either a stream of hexadecimal numbers or as a quoted string of ASCII characters. The length of the generic data is automatically determined and inserted into the proper field(s) of the RFC1048-style boot reply.

tc=*template-host*
This tag indicates a table continuation. Often, many host entries share common values for certain tags (such as domain servers, etc.). Rather than repeatedly specifying these tags, a full specification can be listed for one host entry and shared by others via the **tc** mechanism.

The *template-host* is a dummy host that does not actually exist and never sends boot requests. Information explicitly specified for a host always overrides information implied by a **tc** tag symbol, regardless of its location within the entry. The value of *template-host* can be the hostname or IP address of any host entry previously listed in the configuration file.

Sometimes it is necessary to delete a specific tag after it has been inferred via **tc**. This can be done using the construction FItag@ which removes the effect of *tag*. For example, to completely undo an RFC1034 domain name server specification, use **:ds@:** at an appropriate place in the configuration entry. After removal with @, a tag is eligible to be set again through the **tc** mechanism.

to=*offset*
This tag specifies the client's time zone offset in seconds from UTC. The time *offset* may be either a signed decimal integer or the keyword **auto** which uses the server's time zone offset. Specifying the **to** symbol as a boolean has the same effect as specifying **auto** as its value.

ts=*ip-address-list*
This tag specifies the IP address(es) of RFC868 Time Protocol server(s).

vm=*magic-cookie*
This tag specifies the RFC1048 vendor information magic cookie. The *magic-cookie* may be one of the following keywords: **auto** (indicating that vendor information is determined by the client's request), **rfc1048** (which always forces an RFC1048-style reply), or **cmu** (which always forces a CMU-style reply).

EXAMPLE

An example **/etc/bootptab** file follows:

```
# The first entry is the template for options
# common to all the X terminals.

global.defaults:\
        :bf=C2300A:\
        :hd=/usr/lib/X11/700X/bin:\
        :hn:\
        :ht=ether:\
        :vm=rfc1048:

# Now the actual entries for the individual
# X terminals are listed.
```

(Requires Optional ARPA Services Software)

```
xterm1:\
        :tc=global.defaults:\
        :ha=08000903212F:\
        :ip=190.40.101.22:

xterm2:\
        :tc=global.defaults:\
        :ha=0800090324AC:\
        :ip=190.40.101.35:
```

FILES

/etc/bootptab
/etc/services

WARNINGS

Individual host entries must not exceed 1024 characters.

AUTHOR

bootpd was developed by Carnegie Mellon University and Stanford University.

SEE ALSO

inetd(1m); DARPA Internet Request For Comments RFC951, RFC1048, RFC1084, Assigned Numbers

(Requires Optional ARPA Services Software)

NAME

bootpquery - send BOOTREQUEST to BOOTP server

SYNOPSIS

/etc/bootpquery *haddr* [*htype*] [*options*]

DESCRIPTION

bootpquery is a diagnostic tool used to check the configuration of the Internet Bootstrap Protocol (BOOTP) server, *bootpd*(1M). It uses the supplied parameters to construct a boot request to send to a BOOTP server. It prints the contents of the boot reply, including the client's internet address, the name of a boot file, and the name and address of the server that sent the reply. *bootpquery* formats and prints RFC-1048 or CMU-style vendor information included in the reply. Since it uses reserved ports, it is can only be run by the super-user.

The BOOTREQUEST packet is broadcast on the BOOTP server port **bootps**. If a BOOTP server is configured to respond to the request, it returns a BOOTREPLY packet on the BOOTP client port, **bootpc**. *bootpquery* can only display BOOTREPLY packets when the BOOTP server broadcasts the reply on the client port or when the hardware address and IP address supplied in the BOOTREQUEST are those of the host on which *bootpquery* is run.

The following options provide the information for the BOOTREQUEST:

haddr The hardware adresss of the BOOTP client to use in the BOOTREQUEST. A BOOTP server responds if it has configuration information for a host with this link-level address.

htype The type of address specified as *haddr*, which may be **ether** or **ieee802**. The default address type is **ether**.

-i*ipaddr*
The internet address of the BOOTP client to specify in the BOOTREQUEST. If the BOOTP client doesn't know its IP address, the BOOTP server supplies it in the BOOTREPLY. Otherwise, the server returns the BOOTREPLY directly to ipaddr.

-s*server* The name of the BOOTP server to which the BOOTREQUEST should be sent directly. When the BOOTP server is known, the BOOTREQUEST is not broadcast.

-v*vendor*
Request vendor information for *vendor*. The *vendor* can be specified as **rfc1048** or **cmu**. For any other *vendor* specification, the first four characters of the parameter are used as the vendor magic cookie.

-f*bootfile*
Specify a boot file needed by the BOOTP client. If a boot file is specified in the BOOTREQUEST, the BOOTP server responds only if the server host can make the file available via *tftp*.

EXAMPLE

/etc/bootpquery 02608cee018e ether -s hpserver

Received BOOTREPLY from hpserver.hp.com (15.9.18.119)

Hardware Address:	02:60:8c:ee:01:8e
Hardware Type:	ethernet
IP Address:	15.9.18.113
Boot file:	/usr/tftpdir/hp-gw2-confg

RFC 1048 Vendor Information:

Subnet Mask:	255.255.248.0
Bootfile Size:	6 512 byte blocks
Domain Name Server:	15.9.18.119
Host Name:	hp-gw2

AUTHOR

bootpquery was developed by HP.

SEE ALSO

bootpd(1M), tftp(1), tftpd(1M)
DARPA Internet Request For Comments RFC951, RFC1048, RFC1084, Assigned Numbers.

NAME

brc, bcheckrc, mirrorrc, rc, powerfail - system initialization shell scripts

SYNOPSIS

`/etc/brc`

`/etc/bcheckrc`

`/etc/mirrorrc`

`/etc/rc`

`/etc/powerfail`

Remarks:

The mirror disk features described on this page require installation of optional Data Pair 800 software (not included in the standard HP-UX operating system) before they can be used.

DESCRIPTION

These shell procedures are executed via entries in `/etc/inittab` by `init` (see *init*(1M)). `bcheckrc`, `brc`, and `mirrorrc` are executed when the system is changed out of single-user mode. `rc` is executed upon entering a new, numbered, run-level. `powerfail` is executed whenever a system power failure is detected.

`brc` Clears the mounted file system table `/etc/mnttab` (see *mnttab*(4)), and loads any programmable microprocessors with their appropriate scripts.

`bcheckrc` Performs all necessary consistency checks to prepare the system to change into multi-user mode. It prompts to set the system date and to check the file systems with `fsck` (see *fsck*(1M)).

`mirrorrc` Sets up mirror disk pairs based on information contained in `/etc/mirrortab` (see *mirrortab*(4)). It invokes `mirror` to configure the mirror pairs and, if necessary, to reimage the mirrors for data consistency (see *mirror*(1M)). It is run by `bcheckrc` before `fsck` is run.

`rc` Starts all system daemons before the terminal lines are enabled for multi-user mode. In addition, file systems are mounted, and accounting, error logging, system activity logging, and the Remote Job Entry (RJE) system are activated in this procedure.

`powerfail`
Invoked when the system detects a power-failure condition. Its chief duty is to reload any programmable microprocessors with their appropriate scripts, where applicable. It also logs the fact that a power failure occurred.

SEE ALSO

fsck(1M), init(1M), mirror(1M), shutdown(1M), inittab(4), mirrortab(4), mnttab(4).

NAME

buildlang - generate and display locale.def file

SYNOPSIS

buildlang [**-n**] *input_file*
buildlang **-d** [**-f** *form*] *locale_name*

buildlang is provided for historical reasons only. Use **localedef** instead (see *localedef*(1M)).

DESCRIPTION

Without the **-d** option, **buildlang** automatically sets up the language environment as specified by *input_file*. **buildlang** reads a " buildlang script" (see description below) from *input_file*, creates a file called **locale.def**, and installs this file in the appropriate directory. If the **-n** option is specified, **buildlang** creates the **locale.def** in the current directory.

If the **-d** option is specified, **buildlang** displays the contents of the **locale.def** file associated with the *locale_name* in the format of a "buildlang script" so that the output can be modified and used as an input file to **buildlang** to generate a modified **locale.def** file. If a character code is printable, as defined by the current setting of the **LC_CTYPE** environment variable in the user's environment, **buildlang** always outputs the character code in the "character constant" form. If a character code is not printable, the **-f** option specifies the form in which the character code will be displayed. The **-fc**, **-fd**, **-fo**, and **-fx** options cause **buildlang** to output each non-printable character code in the "character constant", "decimal constant", "octal constant", and "hexadecimal constant" form, respectively (see Constants section below). Without the **-f** option, **buildlang** display each printable character code in the "character constant" form and non-printable character code in the "hexadecimal constant" form.

There are six categories of data in the **locale.def** file, recognized by **setlocale()**, which make up a language definition (see *setlocale*(3C)). They are:

LC_COLLATE	Information in this category affects the behavior of the regular expressions and the NLS string collation functions.
LC_CTYPE	Information in this category affects the behavior of the character classification and conversion functions.
LC_MONETARY	Information in this category affects the behavior of functions which handle monetary values.
LC_NUMERIC	Information in this category affects the handling of the radix character in the formatted input/output functions and the string conversion functions.
LC_TIME	Information in this category affects the behavior of the time conversion functions.
LC_ALL	This category contains language-specific information which does not belong to any of the above categories.

A **buildlang** script also consists of six categories. The beginning of each category is identified by a "category tag" which has the form of **LC_***category* where *category* is one of the following: **ALL**, **COLLATE**, **CTYPE**, **MONETARY**, **NUMERIC**, or **TIME**. The end of each category is identified by a **END_LC** category tag. The order of categories in the "buildlang script" is irrelevant. All category specifications are optional. If a category is not specified, **setlocale()** sets up the default "C" locale for that category (see *setlocale*(3C) and *lang*(5)).

Each category is composed of one or more statements. Each statement begins with a keyword followed by one or more expressions. An expression is a set of well-formed metacharacters, strings, and constants. **buildlang** also recognizes comments and separators.

More than one definition can be specified for each category. If a category contains more than one definition, each additional definition must be named via the **modifier** keyword described below. The first set of specifications is the default definition which may or may not have a modifier name.

The following is a list of category tags, keywords and subsequent expressions which are recognized by **buildlang**. The order of keywords within a category is irrelevant with the exception of the **modifier** keyword. All keyword specifications are optional with the exception of the **langname** and **langid** keywords. (Note that, as a convention, the category tags are composed of uppercase characters, while the

keywords are composed of lowercase characters).

Category Tags and Keywords

The following keywords do not belong to any category.

langname String identifying the name of the language. It follows the naming convention of the **LANG** environment variable:

language[*_territory*][*.codeset*]

(see *environ*(5)). This keyword is required by **buildlang**.

langid Decimal number identifying the language id. This keyword is required by **buildlang**. The language id specified should be in the range of 1 to 999, and any user-defined language should assign its language id in the range of 901 to 999.

revision String identifying the revision number of the **locale.def** file. The string is restricted to contain at most 6 characters, all digits and one optional decimal point (.) character.

The following keyword can be used in any category. It must be used to name a definition when a category contains more than one definition.

modifier String identifying the name of the modifier (see *environ*(5)). Since this keyword is used to associate a modifier with a set of specifications, it must come before any keyword in that set of specifications.

LC_ALL:

The following keywords belong to the **LC_ALL** category and should come between the category tag **LC_ALL** and **END_LC**:

yesstr String identifying the affirmative response for yes/no questions (a *langinfo*(5) item).

nostr String identifying the negative response for yes/no questions (a *langinfo*(5) item).

direction String indicating text direction (a *langinfo*(5) item).

context String indicating character context analysis. String "null" or "0" indicates no context analysis is required. String "1" indicates Arabic context analysis required.

LC_CTYPE:

The following keywords belong to the **LC_CTYPE** category and should come between the category tag **LC_CTYPE** and **END_LC**:

isupper Character codes classified as uppercase letters.

islower Character codes classified as lowercase letters.

isdigit Character codes classified as numeric.

isspace Character codes classified as spacing (delimiter) characters.

ispunct Character codes classified as punctuation characters.

iscntrl Character codes classified as control characters.

isblank Character codes for printable space characters. These also must be defined in **isspace**.

isxdigit Character codes classified as hexadecimal digits.

isfirst Character codes classified as the first bytes of two-byte characters.

issecond Character codes classified as the second bytes of two-byte characters.

ul Relationships between uppercase and lowercase characters. Used for languages that have a one-to-one relationship between lowercase and uppercase characters.

toupper Lowercase to uppercase character relationships.

tolower Uppercase-to-lowercase character relationships. Keywords **toupper** and **tolower** are used only for languages that do not have a one-to-one relationship between lowercase and uppercase characters.

bytes_char String containing the maximum number of bytes per character for the character set used for a specified language (a *langinfo*(5) item).

alt_punct String mapped into the ASCII equivalent string **b!"#$%&'()*+,-./:;<=>?@[\]^_'{~**, where b is a blank (a *langinfo*(5) item).

code_scheme
Specifies the multi-byte character encoding scheme used. The operand should be a string. Currently, "HP15" and "EUC" strings are recognized. If this keyword is not specified, or the operand is a null string (f3""), the encoding scheme is single-byte, or HP15 if **bytes_char** is **2**. See *Native Language Support* User's Guide.

cswidth Defines the number of bytes contained in a character, and the number of columns per character displayed on output devices. This keyword should be specified if the encoding scheme is "EUC". EUC can be divided into 4 Supplementary Code Sets. The first SCS, Supplementary Code Set 0, contains ASCII characters and is assumed to contain 1 byte per character and require 1 column on the output devices. The operand is a string containing three ordered pairs of digits delimited by colons and commas. The format is:

X:*x*,*Y*:*y*,*Z*:*z*

Field	Interpretation
X	SCS 1, number of bytes
x	SCS 1, output width
Y	SCS 2, number of bytes, after SS2
y	SCS 2, output width
Z	SCS 3, number of bytes, after SS3
z	SCS 3, output width

LC_COLLATE:
The following keywords belong to the **LC_COLLATE** category and should come between the category tag **LC_COLLATE** and **END_LC**:

sequence Sequence of character codes for collation.

LC_MONETARY:
The following keywords belong to the **LC_MONETARY** category and should come between the category tag **LC_MONETARY** and **END_LC**. These keywords, except **crncystr**, are identical to the members in struct *lconv* defined in **<locale.h>** (see *localeconv*(3)):

int_curr_symbol
currency_symbol
mon_decimal_point
mon_thousands_sep
mon_grouping
positive_sign
negative_sign
int_frac_digits
frac_digits
p_cs_precedes
p_sep_by_space
n_cs_precedes
n_sep_by_space
p_sign_posn
n_sign_posn

crncystr String for specifying the currency (a *langinfo*(5) item).

LC_NUMERIC:
The following keywords belong to the **LC_NUMERIC** category and should come between the category tag **LC_NUMERIC** and **END_LC**: These keywords, except **alt_digits**, are identical to the members in struct **lconv** defined in **<locale.h>** (see *localeconv*(3C)).

`grouping`

`decimal_point`
(same as **RADIXCHAR**, a *langinfo*(5) item).

`thousands_sep`
(same as **THOUSEP**, a *langinfo*(5) item).

`alt_digits` String mapped into the ASCII equivalent string `0123456789`*b*`+-.,eE`, where *b* is a blank (a *langinfo*(5) item).

`LC_TIME:`
The following keywords belong to the `LC_TIME` category and should come between the category tag `LC_TIME` and `END_LC`: These keywords, except `era`, are identical to their corresponding *langinfo*(5) items (see *langinfo*(5)).

`d_t_fmt`
`d_fmt`
`t_fmt`
`day_1` to `day_7`
`abday_1` to `abday_7`
`mon_1` to `mon_7`
`abmon_1` to `abmon_7`
`am_str`
`pm_str`
`year_unit`
`mon_unit`
`day_unit`
`hour_unit`
`min_unit`
`sec_unit`
`era_fmt`

`era` Names and dates of eras or emperors.

Expressions
Expressions consist of character-code constants, strings, and metacharacters. There are four types of legal expressions; `ctype`, `shift`, `collate`, and `info`:

`ctype` `Ctype` expressions follow the keywords `isupper`, `islower`, `isdigit`, `isspace`, `ispunct`, `iscntrl`, `isblank`, `isxdigit`, `isfirst`, and `issecond` and can include either a single character-code constant or a character-code range consisting of a constant followed by a dash followed by another constant. At least one separator must appear between the constants and dash. The constant preceding the dash must have a smaller code value than the constant following the dash. A range represents a set of consecutive character codes.

`shift` `Shift` expressions follow keywords `ul`, `toupper`, and `tolower`, and must consist of two character-code constants enclosed by left and right angle brackets. For `ul` and `tolower`, the first constant represents an uppercase character and the second the corresponding lowercase character. For `toupper`, the first constant represents an lowercase character and the second the corresponding uppercase character.

`collate`

`Collate` expressions that follow the keyword `sequence` represent a sequence of character codes that define a collation order. Each character code in the series is assigned an ascending sequence number. `Collate` expressions include single character-code constants, character-code ranges, character-code priority sets, two-to-one character-code pairs, one-to-two character-code pairs and character-code don't-care sets.

A character-code priority set is a collection of one or more constants or other `collate` expressions enclosed by left and right parenthesis. Constants or expressions within a priority set havé the same collation sequence number but different priorities. The priorities account for case and accent differences.

A two-to-one character-code pair is represented by two character-code constants enclosed by left and right angle brackets. Two-to-one characters are two adjacent characters that occupy one position in the collating sequence. For example, the expression sequence `('C' 'c') ( <'C' 'h'> <'c' 'h'>) ('D' 'd')` instructs **buildlang** to treat the character combinations `Ch` and `ch` as single characters that collate between lowercase `c` and uppercase `D`.

A one-to-two character code is represented by two character-code constants enclosed by left and right brackets. One-to-two characters are single characters that occupy two adjacent positions in the collating sequence. For example, suppose the character 'X' represents a one-to-two character that collates as **AE**. This information can be expressed as `('A' ['X' 'E'] 'a')`. The character 'X' has the same primary sequence number as 'A' and 'a', a priority that lies between 'A' and 'a' and a secondary sequence number that is the same as 'E'.

A character-code don't-care set is a collection of one or more constants or other collate expressions enclosed by left and right curly brackets. Constants or expressions within a don't-care set are ignored in character comparisons.

`info` `Info` expressions follow all *langinfo*-type, *lconv*-type and `era` keywords. Each expression is a string (see `Strings` section below).

The expressions following the *langinfo*-type keywords define the strings associated with *items* in *langinfo*(5). Each expression consists of a string to be associated with the *item* identified by the keyword.

The expressions following the `lconv`-type keywords define the strings associated with *members* of `lconv` struct in *localeconv*(3C). Each expression consists of a string to be associated with the *member* identified by the keyword.

Each expression following the keyword `era` defines how the years are counted and displayed for one era (or emperor's reign). The expressions must be in the following format:

direction`:`*offset*`:`*start_date*`:`*end_date*`:`*name*`:`*format*

where:

direction Either a `+` or `-` character. The `+` character indicates the time axis should be such that the years count in the positive direction when moving from the starting date towards the ending date. The `-` character indicates the time axis should be such that the years count in the negative direction when moving from the starting date towards the ending date.

offset A number in the range `[SHRT_MIN,SHRT_MAX]` indicating the number of the first year of the era.

start_date A date in the form *yyyy*`/`*mm*`/`*dd* where *yyyy*, *mm*, and *dd* are the year, month and day numbers, respectively, of the start of the era. Years prior to the year 0 A.D. are represented as negative numbers. For example, an era beginning March 5th in the year 100 B.C. would be represented as `-100/3/5`. Years in the range `[SHRT_MIN+1,SHRT_MAX-1]` are supported.

end_date The ending date of the era in the same form as the start_date above or one of the two special values `-*` or `+*`. A value of `-*` indicates that the ending date of the era extends to the beginning of time while `+*` indicates it extends to the end of time. The ending date can be chronologically either before or after the starting date of an era. For example, the expressions for the Christian eras A.D. and B.C. would be:

```
+:0:0000/01/01:+*:A.D.:%o %N
+:1:-0001/12/31:-*:B.C.:%o %N
```

name A string representing the name of the era which is substituted for the `%N` directive of *date*(1) and *strftime*(3C).

format A string for formatting the **%E** directive of *date*(1) and *strftime*(3C). This string is usually a function of the **%O** and **%N** directives. If *format* is not specified, the string specified for the **LC_TIME** category keyword **era_fmt** (see above) is used as a default.

Constants

Constants represent character codes in the **ctype**, **shift** and **collate** expressions. C programming language integer and character constants can be used as character codes, including:

decimal constants	Sequence of digits not beginning with a **0** (zero).
octal constants	Sequence of digits beginning with a **0** (zero).
hexadecimal constants	Sequence of digits preceded by **0x** or **0X** (zero followed by character x or X). Hexadecimal digits include **a** or **A** through **f** or **F**, with values **10** through **15**.
character constants	A single character written between single quotes, having the numerical value of the character in the machine's character set.

Strings

Strings are used in **info** expressions. A string is a sequence of zero or more characters surrounded by double quotes ("). Within a string, the double-quote character must be preceded by a backslash (\). The following escape sequences also can be used:

\n	newline
\t	horizontal tab
\b	backspace
\r	carriage return
\f	form feed
****	backslash
\'	single quote
\ddd	bit pattern

The escape *\ddd* consists of the backslash followed by 1, 2, or 3 octal digits specifying the value of the desired character. Also, a backslash (\) and an immediately-following newline are ignored.

Metacharacters

Metacharacters are characters having a special meaning to **buildlang** in **ctype**, **shift**, and **collate** expressions. To escape the special meaning of these characters, surround them with single quotes. **buildlang** meta-characters include:

-	Represents a range of consecutive character codes.
<	When used with the **ul**, **toupper**, and **tolower** keywords, indicates the beginning of an uppercase-lowercase character code relationship. When used with the **sequence** keyword, indicates the beginning of a two-to-one character pair.
>	When used with the **ul**, **toupper**, and **tolower** keywords, indicates the end of an uppercase-lowercase character code relationship. When used with the **sequence** keyword, indicates the end of a two-to-one character pair.
[	Indicates the beginning of a one-to-two character pair.
]	Indicates the end of a one-to-two character pair.
(	Indicates the beginning of a group of character code constants or expressions having the same collation sequence number, but different priorities.
)	Indicates the end of a group of character code constants or expressions having the same collation sequence number, but different priorities.
{	Indicates the beginning of a group of character code constants or expressions belonging to the same set of collation don't-care characters.

} Indicates the end of a group of character code constants or expressions belonging to the same set of collation don't-care characters.

Comments
Comments are all characters between a pound sign (#) and a carriage return, except when used in the character code constants and strings. Comments and blank lines are ignored.

Separators
Separator characters include blanks and tabs. Any number of separators can be used to delimit the keywords, metacharacters, constants and strings that comprise a `buildlang` script.

EXTERNAL INFLUENCES

Environment Variables
`LC_CTYPE` determines the printable characters when the `-d` option is specified.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LC_CTYPE`.

International Code Set Support
Single- and multi-byte character code sets are supported.

EXAMPLES

The following `buildlang` script creates the `locale` file for the `american` language using the ROMAN8 code set:

```
# language: american # code set: ROMAN8

langname        "american"
langid  1
revision        "1.1"

##################################################
# Set up the LC_ALL category of the table

LC_ALL
yesstr  "yes"    # yes string
nostr   "no"     # no string
direction        ""      # left-to-right orientation
context ""
END_LC

##################################################
# Set up the LC_CTYPE category of the table

LC_CTYPE
isupper 'A' - 'Z'                # true if an uppercase char
        0xa1 - 0xa7      0xad  0xae       0xb1  0xb4  0xb6
        0xd0  0xd2       0xd3  0xd8       0xda - 0xdc
        0xde - 0xe1      0xe3    0xe5 - 0xe9
        0xeb  0xed       0xee  0xf0
islower 'a' - 'z'                # true if a lowercase char
        0xb2  0xb5       0xb7    0xc0 - 0xcf  0xd1
        0xd4 - 0xd7      0xd9  0xdd       0xde  0xe2
        0xe4  0xea       0xec  0xef       0xf1
isdigit '0' - '9'                # true if a digit
isspace ' '      0x9 - 0xd       # true if a space
ispunct '!' - '/'        ':' - '@'       # true if a punctuation char
        '[' - '`'        '{' - '~'
        168 - 172        175 176 179
        184 - 191        242 - 254
iscntrl 0x0 - 0x1f       0x7f    # true if a control char
        128 - 160        255
isblank ' '              # true if a blank char
isxdigit         '0' - '9'       'a' - 'f'       # true if a hex digit
                 'A' - 'F'
```

```
# isfirst and issecond are irrelevant here
ul      < 'A' 'a' >     < 'B' 'b' >     # < upper  lower>
        < 'C' 'c' >     < 'D' 'd' >
        < 'E' 'e' >     < 'F' 'f' >
        < 'G' 'g' >     < 'H' 'h' >
        <0x49 0X69>     <0x4a 0X6a>     # hex constants allowed
        <0113 0153>     <0114 0154>     # octal constants allowed
        < 77  109 >     < 78  110 >     # decimal constants allowed
        < 'O' 'o' >     < 'P' 'p' >     # literal constants allowed
        < 'Q' 'q' >     < 'R' 'r' >
        < 'S' 's' >     < 'T' 't' >
        < 'U' 'u' >     < 'V' 'v' >
        < 'W' 'w' >     < 'X' 'x' >
        < 'Y' 'y' >     < 'Z' 'z' >
        <0xa1 0xc8>     <0xa2 0xc0>
        <0xa3 0xc9>     <0xa4 0xc1>
        <0xa5 0xcd>     <0xa6 0xd1>
        <0xa7 0xdd>     <0xad 0xcb>
        <0xae 0xc3>     <0xb4 0xb5>
        <0xb6 0xb7>     <0xd0 0xd4>
        <0xd2 0xd6>     <0xd3 0xd7>
        <0xd8 0xcc>     <0xda 0xce>
        <0xdb 0xcf>     <0xdc 0xc5>
        <0xdf 0xc2>     <0xe0 0xc4>
        <0xe1 0xe2>     <0xe3 0xe4>
        <0xe5 0xd5>     <0xe6 0xd9>
        <0xe7 0xc6>     <0xe8 0xca>
        <0xe9 0xea>     <0xeb 0xec>
        <0xed 0xc7>     <0xee 0xef>
        <0xf0 0xf1>
bytes_char      "1"     # max number of bytes per char
alt_punct       ""      # no alternative punctuation
code_scheme     ""      # encoding scheme is single-byte
END_LC

##################################################
# Set up the LC_COLLATE category of the table

# dictionary collating sequence:
# spaces, decimal digits,
# alphabetic characters, punctuation,
# control characters

LC_COLLATE
modifier        "nofold"
sequence        ' ' 0xa0 '0' - '9'
        ( 'A' [0xd3 'E'] 0xe0 0xa1 0xa2 0xd8 0xd0 0xe1 ) 'B'
        ( 'C' 0xb4 ) ( 'D' 0xe3 ) ( 'E' 0xdc 0xa3 0xa4 0xa5 ) 'F'
        'G' 'H' ( 'I' 0xe5 0xe6 0xa6 0xa7 ) 'J' 'K' 'L' 'M'
        ( 'N' 0xb6 ) ( 'O' 0xe7 0xe8 0xdf 0xda 0xe9 0xd2 ) 'P' 'Q'
        'R' ( 'S' 0xeb ) 'T' ( 'U' 0xed 0xad 0xae 0xdb )
        'V' 'W' 'X' ( 'Y' 0xee ) 'Z' 0xf0
        ( 'a' [0xd7 'e'] 0xc4 0xc8 0xc0 0xcc 0xd4 0xe2 ) 'b'
        ( 'c' 0xb5 ) ( 'd' 0xe4 ) ( 'e' 0xc5 0xc9 0xc1 0xcd ) 'f'
        'g' 'h' ( 'i' 0xd5 0xd9 0xd1 0xdd) 'j' 'k' 'l' 'm'
        ( 'n' 0xb7 ) ( 'o' 0xc6 0xca 0xc2 0xce 0xea 0xd6 ) 'p' 'q'
        'r' ( [0xde 's'] 's' 0xec ) 't' ( 'u' 0xc7 0xcb 0xc3 0xcf )
        'v' 'w' 'x' ( 'y' 0xef ) 'z' 0xf1
        0xb1  0xb2  0xf2 - 0xf5  '(' ')' '[' ']'
        '{' '}'  0xfb  0xfd  '<' '>' '=' '+' '-'  0xfe  0xf7  0xf8
```

```
        0xb3  '%' '*' '.' ',' ';' ':'  0xb9  '?'  0xb8  '!'
        '/' '\\'  '|' '@' '&' '#'  0xbd  '$'  0xbf  0xbb  0xaf
        0xbc  0xbe  0xba  '"' ''' ''' '^' '~'  0xa8 - 0xac  '_'
        0xf6  0xb0  0xf9  0xfa  0xfc  0x0 - 0x1f  0x80 - 0x9f
        0x7f  0xff

modifier        "fold"
sequence        ' ' 0xa0 '0' - '9'
        ( 'A' [0xd3 'E'] 'a' [0xd7 'e'] 0xe0 0xc4 0xa1 0xc8 0xa2 0xc0
          0xd8 0xcc 0xd0 0xd4 0xe1 0xe2 )
        ( 'B' 'b' ) ( 'C' 'c' 0xb4 0xb5 ) ( 'D' 'd' 0xe3 0xe4 )
        ( 'E' 'e' 0xdc 0xc5 0xa3 0xc9 0xa4 0xc1 0xa5 0xcd ) ( 'F' 'f' )
        ( 'G' 'g' ) ( 'H' 'h' )
        ( 'I' 'i' 0xe5 0xd5 0xe6 0xd9 0xa6 0xd1 0xa7 0xdd) ( 'J' 'j' )
        ( 'K' 'k' ) ( 'L' 'l' ) ( 'M' 'm' ) ( 'N' 'n' 0xb6 0xb7 )
        ( 'O' 'o' 0xe7 0xc6 0xe8 0xca 0xdf 0xc2 0xda 0xce 0xe9 0xea
          0xd2 0xd6 ) ( 'P' 'p' ) ( 'Q' 'q' ) ( 'R' 'r' )
        ( 'S' [0xde 's'] 's' 0xeb 0xec ) ( 'T' 't' )
        ( 'U' 'u' 0xed 0xc7 0xad 0xcb 0xae 0xc3 0xdb 0xcf ) ( 'V' 'v' )
        ( 'W' 'w' ) ( 'X' 'x' ) ( 'Y' 'y' 0xee 0xef ) ( 'Z' 'z' )
        ( 0xf0 0xf1 ) 0xb1  0xb2  0xf2 - 0xf5  '(' ')' '[' ']'
        '{' '}'  0xfb  0xfd  '<' '>' '=' '+' '-'  0xfe  0xf7  0xf8
        0xb3  '%' '*' '.' ',' ';' ':'  0xb9  '?'  0xb8  '!'
        '/' '\\'  '|' '@' '&' '#'  0xbd  '$'  0xbf  0xbb  0xaf
        0xbc  0xbe  0xba  '"' ''' ''' '^' '~'  0xa8 - 0xac  '_'
        0xf6  0xb0  0xf9  0xfa  0xfc  0x0 - 0x1f  0x80 - 0x9f
        0x7f  0xff
END_LC

##################################################
# Set up the LC_MONETARY category of the table

LC_MONETARY
int_curr_symbol "USD "
currency_symbol "$"
mon_decimal_point       "."
mon_thousands_sep       ","
mon_grouping    "\\3"
positive_sign   ""
negative_sign   "-"
int_frac_digits "2"
frac_digits     "2"
p_cs_precedes   "1"
p_sep_by_space  "0"
n_cs_precedes   "1"
n_sep_by_space  "0"
p_sign_posn     "1"
n_sign_posn     "4"
crncystr        "-US$"
END_LC

##################################################
# Set up the LC_NUMERIC category of the table

LC_NUMERIC
grouping        "\\3"
thousands_sep   ","     # THOUSEP: thousands separator
decimal_point   "."     # RADIXCHAR: radix character
alt_digits      ""      # no alternative digits
END_LC

##################################################
```

```
# Set up the LC_TIME category of the table

LC_TIME
d_t_fmt "%a, %h %d, 19%y %r"    # date & time format string
d_fmt   "%a, %h %d, 19%y"       # date format string
t_fmt   "%r"    # time format string
day_1   "Sunday"        # weekday names
day_2   "Monday"
day_3   "Tuesday"
day_4   "Wednesday"
day_5   "Thursday"
day_6   "Friday"
day_7   "Saturday"
abday_1 "Sun"   # weekday abbreviations
abday_2 "Mon"
abday_3 "Tue"
abday_4 "Wed"
abday_5 "Thu"
abday_6 "Fri"
abday_7 "Sat"
mon_1   "January"       # month names
mon_2   "February"
mon_3   "March"
mon_4   "April"
mon_5   "May"
mon_6   "June"
mon_7   "July"
mon_8   "August"
mon_9   "September"
mon_10  "October"
mon_11  "November"
mon_12  "December"
abmon_1 "Jan"   # month abbreviations
abmon_2 "Feb"
abmon_3 "Mar"
abmon_4 "Apr"
abmon_5 "May"
abmon_6 "Jun"
abmon_7 "Jul"
abmon_8 "Aug"
abmon_9 "Sep"
abmon_10        "Oct"
abmon_11        "Nov"
abmon_12        "Dec"
am_str  "AM"    # AM string
pm_str  "PM"    # PM string
year_unit       ""      # the unit of year
mon_unit        ""      # the unit of month
day_unit        ""      # the unit of day
hour_unit       ""      # the unit of hour
min_unit        ""      # the unit of minute
sec_unit        ""      # the unit of second

# There is no era or emperor year for the american language,
# but here is an example of the japanese era_fmt and era specification:

era_fmt "%N%onen"       # normal era format string
era     "+:2:1990/01/01:+*:Heisei"
        "+:1:1989/01/08:1989/12/31:Heisei:%Ngannen"     # special format for 1st year
        "+:2:1927/01/01:1989/01/07:Shouwa"
        "+:1:1926/12/25:1926/12/31:Shouwa:%Ngannen"     # special format for 1st year
```

```
            "+:2:1913/01/01:1926/12/24:Taishou"
            "+:1:1912/07/30:1912/12/31:Taishou:%Ngannen"    # special format for 1st year
            "+:2:1869/01/01:1912/07/29:Meiji"
            "+:1:1868/09/08:1868/12/31:Meiji:%Ngannen"      # special format for 1st year
            "-:1868:1868/09/07:-*::%o"      # revert to regular year numbering for
                    # years prior to the supported eras
END_LC
```

ERRORS

If **buildlang** detects any errors, it terminates with an error message and will not generate a **locale.def** file.

WARNINGS

To specify a 16-bit character, it is recommended to use two bit-pattern (*\ddd*) escape sequences.

AUTHOR

buildlang was developed by HP.

FILES

/usr/lib/nls/config
/usr/lib/nls/*language*[**/***territory*][**/***codeset*]**/locale.def**

SEE ALSO

setlocale(3C), environ(5).

NAME

captoinfo - convert a termcap description into a terminfo description

SYNOPSIS

captoinfo [**-1v**] [**-w***n*] [*filenames*]

DESCRIPTION

captoinfo looks in *filenames* for *termcap*(3X) descriptions. For each one found, an equivalent *terminfo*(4) description is written to standard output along with any comments found. The short two letter name at the beginning of the list of names in a *termcap* entry, a hold-over from Version 6 UNIX , is removed. Any description that is expressed relative to another description (as specified in the termcap *tc*= field) is reduced to the minimum superset before output.

If no *filename* is given, the environment variable TERMCAP is used for the filename or entry. If TERMCAP is a full pathname to a file, only the terminal whose name is specified in the environment variable TERM is extracted from that file. If the environment variable TERMCAP is not set, the file **/etc/termcap** is read.

Options

captoinfo recognizes the following options:

-1 Print one field per line. If this option is not selected, multiple fields are printed on each line up to a maximum width of 60 characters.

-v Print (verbose) tracing information as the program runs. Additional **-v** options print more information (for example **-v -v -v** or **-vvv**).

-w*n* Change the output width to *n* characters.

WARNINGS

Certain *termcap* defaults are assumed to be true. For example, the bell character (terminfo *bel*) is assumed to be ^*G*. The linefeed capability (termcap *nl*) is assumed to be the same for both *cursor_down* and *scroll_forward* (terminfo *cud1* and *ind*, respectively.) Padding information is assumed to belong at the end of the string.

The algorithm used to expand parameterized information for *termcap* fields such as *cursor_position* (termcap *cm*, terminfo *cup*) sometimes produces a string which, though technically correct, may not be optimal. In particular, the rarely used termcap operation **%n** produces strings that are especially long. Most occurrences of these non-optimal strings are flagged with a warning message, and may need to be recoded by hand.

HP supports only terminals listed on the current list of supported devices. However, the terminfo database contains both supported and non-supported terminals. If you use non-supported terminals, they may not work correctly.

DIAGNOSTICS

tgetent failed with return code n (reason).
The termcap entry is not valid. In particular, check for an invalid 'tc=' entry.

unknown type given for the termcap code 'cc'.
The termcap description had an entry for 'cc' whose type was not boolean, numeric or string.

wrong type given for the boolean (numeric, string) termcap code 'cc'.
The boolean termcap entry 'cc' was entered as a numeric or string capability.

the boolean (numeric, string) termcap code 'cc' is not a valid name.
An unknown termcap code was specified.

tgetent failed on TERM=term.
The terminal type specified could not be found in the termcap file.

TERM=term: cap cc (info ii) is NULL: REMOVED
The termcap code was specified as a null string. The correct way to cancel an entry is with an '@', as in ':bs@:'. Giving a null string could cause incorrect assumptions to be made by any software that uses termcap or terminfo.

a function key for 'cc' was specified, but it already has the value 'vv'.
When parsing the 'ko' capability, the key 'cc' was specified as having the same value as the capability 'cc', but the key 'cc' already had a value assigned to it.

the unknown termcap name 'cc' was specified in the 'ko' termcap capability.
A key that could not be handled was specified in the 'ko' capability.

the vi character 'v' (info 'ii') has the value 'xx', but 'ma' gives 'n'.
The 'ma' capability specified a function key with a value different from that specified in another setting of the same key.

the unknown vi key 'v' was specified in the 'ma' termcap capability.
A vi key unknown to *captoinfo* was specified in the 'ma' capability.

Warning: termcap sg (nn) and termcap ug (nn) had different values.
Terminfo assumes that the sg (now xmc) and ug values were the same.

Warning: the string produced for 'ii' may be inefficient.
The parameterized string being created should be rewritten by hand.

Null termname given.
The terminal type was null. This occurs when $TERM is null or not set.

cannot open %s" for reading."
The specified file could not be opened.

Warning: cannot translate <capability> **(unsupported in terminfo).**
This termcap capability is no longer supported in terminfo, and therefore cannot be translated.

AUTHOR
captoinfo was developed by AT&T.

SEE ALSO
curses (3X), termcap (3X), terminfo (4), tic (1M), untic (1M).

NAME

catman - create the cat files for the manual

SYNOPSIS

/etc/catman [**-p**] [**-m**] [**-n**] [**-w**] [**-z**] [*sections*]

DESCRIPTION

catman creates the formatted versions of the online manual from *nroff*(1)-compatible source files. Each manual entry in the **man*.Z** and **man*** directories is examined, and those whose formatted versions are missing or out-of-date are recreated. *catman* formats the most recent of the entries, compresses it, and puts it into the appropriate **cat*.Z** directory.

If any changes are made, *catman* recreates the **/usr/lib/whatis** database. By default, the **/usr/lib/whatis** database is overwritten. If the MANPATH environment varialbe is set to a non-default set of paths, the old database file is saved in **/usr/lib/whatis.old** so that, if desired, the system administrator may merge them together.

By default, *catman* searches the **man*.Z** and **man*** subdirectories under directory paths **/usr/man**, **/usr/contrib/man**, and **/usr/local/man**. If MANPATH is set in the environment, the directories given in MANPATH are checked instead of the default. See *environ(5)* for a description of the MANPATH environment variable.

Before running *catman*, remove any existing **cat*** directories. If the **-z** option is used, **cat*.Z** directories should be removed instead. If both **cat*.Z** and **cat*** directories exist, *man*(1) updates both directories and more space is used.

Any command-line parameters not starting with **-** are interpreted as a list of manual sections (directories) to search. For example:

catman 123

restricts updating to manual sections 1, 2, and 3 (directories **man1**, **man2**, and **man3**).

Options

-m Create a merged **/usr/lib/whatis** database; i.e., information on new manual entries (added since the last time *catman* was run) is merged into the current database rather that overwriting it. Ignored if selected with the **-n** option.

-n Prevents creation of **/usr/lib/whatis**.

-p Prints what would be done instead of doing it.

-w Causes only the **/usr/lib/whatis** database to be created. No manual reformatting is done.

-z Puts the formatted entries in the **cat*** directories rather than in the **cat*.Z** directories.

EXTERNAL INFLUENCES

Environment Variables

MANPATH defines parent directories to be used when searching **man*** and **man*.Z** directories.

WARNINGS

If unformatted manual entries (those in the **../man?** subdirectories) have been removed since the last time *catman* was run, information in the **/usr/lib/whatis** database may be lost. The **-m** option may be used to override this, but may result in repeated lines in the database for the same manual entry.

AUTHOR

catman was developed by HP and the University of California, Berkeley.

FILES

/usr/man/man*[.Z]/*	unformatted (*nroff*(1)-compatible source) manual entry files [compressed]
/usr/man/cat*[.Z]/*	formatted manual pages [compressed]
/usr/local/man/man*[.Z]/*	
/usr/local/man/cat*[.Z]/*	
/usr/contrib/man/man*[.Z]/*	
/usr/contrib/man/cat*[.Z]/*	
/usr/lib/mkwhatis	commands to make whatis database

SEE ALSO

man(1), compress(1), fixman(1), environ(5).

NAME

ccck - HP Cluster configuration file checker

SYNOPSIS

/etc/ccck [*file*]

DESCRIPTION

ccck scans the cluster configuration file and notes any inconsistencies. Checks made include validation of the cluster node machine ID, the number of fields per line, the cluster node name, the cluster node ID, the cluster node type, the swap server field, and the number of cluster server processes.

file specifies the file to check. If *file* is not specified, **/etc/clusterconf** is used.

AUTHOR

ccck was developed by HP.

FILES

/etc/clusterconf

SEE ALSO

clusterconf(4).

NAME

chroot - change root directory for a command

SYNOPSIS

`/etc/chroot` *newroot command*

DESCRIPTION

The specified command is executed *relative to the new root*. The meaning of any initial slashes (`/`) in path names is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Note that:

```
chroot newroot command >x
```

creates file **x** relative to the original root, not the new one.

command includes both the command name and any arguments.

This command is restricted to users with appropriate privileges.

The new root path name is always relative to the current root. Even if a `chroot` is currently in effect, the *newroot* argument is relative to the current root of the running process.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

command cannot be in a shell script.

Exercise extreme caution when referencing special files in the new root file system.

`chroot` does not search **PATH** for the location of *command*, so the absolute path name of *command* must be given.

When using `chroot` to establish a new environment, all absolute pathname references to the file system are lost, rendering shared libraries inaccessible. If continued access to shared libraries is needed for correct operation, the shared libraries and the dynamic loader *must* be copied into the new `chroot`ed environment.

SEE ALSO

chdir(2), chroot(2).

STANDARDS CONFORMANCE

`chroot`: SVID2, XPG2, XPG3

NAME

clri - clear inode

SYNOPSIS

`/etc/clri` *file-system i-number ...*

DESCRIPTION

`clri` writes zeros on the inode numbered *i-number*. *file-system* must be a special file name referring to a device containing a file system. For proper results, *file-system* should not be mounted. After `clri` is executed, any blocks in the affected file show up as "missing" in an `fsck` of *file-system* (see *fsck*(1M)). This command should only be used in emergencies, and extreme care should be exercised.

Read and write permission is required on the specified *file-system* device. The inode becomes allocatable.

WARNINGS

The primary purpose of this command is to remove a file which for some reason appears in no directory. If it is used to zero out an inode that does appear in a directory, care should be taken to locate the entry and remove it. Otherwise, when the inode is reallocated to some new file, the old entry will still point to that file. At that point, removing the old entry destroys the new file, causing the new entry to point to an again unallocated inode with the likelihood that the whole cycle will be repeated.

If the file system is mounted, `clri` is likely to be ineffective.

SEE ALSO

fsck(1M), fsdb(1M), ncheck(1M), fs(4).

STANDARDS CONFORMANCE

clri: SVID2

NAME

clrsvc - clear x25 switched virtual circuit

SYNOPSIS

`clrsvc` *line pad-type*

DESCRIPTION

`clrsvc` clears any virtual circuit that might be established on the specified *line*. *pad-type* indicates to `clrsvc` what `opx25` script to run from `/usr/lib/uucp/X25`.

DEPENDENCIES

HP2334A is the only PAD supported at this time, and results in an `opx25` execution of `HP2334A.clr`.

EXAMPLES

A typical invocation is:

```
/usr/lib/uucp/X25/clrsvc /dev/x25.1 HP2334A
```

AUTHOR

`clrsvc` was developed by HP.

SEE ALSO

getx25(1M), opx25(1M), getty(1M), login(1), uucp(1).

NAME

cluster - allocate resources for clustered operation

SYNOPSIS

`/etc/cluster`

DESCRIPTION

`cluster` allocates kernel resources and notifies the kernel that operation of an HP Cluster is intended. `cluster` can only be executed by the machine listed as the cluster server in the `/etc/clusterconf` file. The machine executing the `cluster` command is converted from a stand-alone system to the cluster server of an HP Cluster. Once the `cluster` command has been executed, the cluster server processes started, and the network initialized, the remote boot daemon can be started, thus allowing other machines to join the cluster. Machines that are allowed to join a cluster are listed in the `/etc/clusterconf` file.

The `cluster` command is normally executed as part of the `/etc/rc` script.

DIAGNOSTICS

`Not listed as root server in /etc/clusterconf`

A machine other than the cluster server attempted to execute this command.

AUTHOR

`cluster` was developed by HP.

FILES

`/etc/clusterconf`

SEE ALSO

csp(1M), ifconfig(1M), rbootd(1M), clusterconf(4).

NAME

config - configure an HP-UX system

SYNOPSIS

`/etc/config` [`-t`] [`-m` *master*] [`-c` *c_file*] [`-l` *m_file*] *dfile*
`/etc/config -a`

DESCRIPTION

`config` is used to configure the following parts of the operating system:

- Device drivers
- Root and swap devices
- Selected system parameters
- Kernel code that handles messages, semaphores, and shared memory

In the first form shown above, `config` reads a user-provided description of an HP-UX system (*dfile*) and generates two output files:

- A C program source file that defines the configuration tables for various parts of the system.
- A makefile (see *make*(1)) to compile the C program produced and relink the newly configured system.

Command Line Arguments

The `config` command can be used in two forms and recognizes the following arguments:

First Form:

`-t` Give a short table of major device numbers for the character and block devices named in *dfile*. This can facilitate the creation of special files.

`-m` *master* Specify that the file *master* contains all the information regarding supported devices. The default file name is `/etc/master`.

`/etc/master` is supplied as part of the HP-UX operating system and should not be modified by anyone who does not fully understand its structure and purpose.

`-c` *c_file* Specify the name of the C program file produced by `config`. The default file name is `conf.c`.

`-l` *m_file* Specify the name of the makefile for compiling the configuration program and relinking the newly configured system. The default file name is `config.mk`.

dfile Specify the name of a file containing configuration information for the user's system. This file is divided into two parts: The first part (mandatory) contains driver specifications; the second part (optional) contains system-dependent information. In this file, any line with an asterisk (`*`) in column 1 is a comment.

Second Form:

`-a` Produce a script of `mknod` templates, placing the result in `./mkdev` (see *mknod*(1M) and *mkdev*(1M)).

The `-a` option cannot be used if *dfile* is specified. If the `-a` option is not specified, *dfile* is required.

Constructing `dfile`: First Part

dfile consists of two parts; the first part is required, whereas the second is optional. Any line in *dfile* that starts with an asterisk (`*`) in column 1 is treated as a comment. The first part of *dfile* is used to configure:

- Device drivers
- Pseudo-drivers, such as *pty*s

Each line has the following format:

devname

where *devname* is the driver name for the device as it appears in the alias table in file `/etc/master`. For example, `scsi` selects the driver for SCSI disk drives, `scsitape` selects the driver for SCSI tape drives, and `dskless` selects the diskless (HP Clustered Environment) protocol. `/etc/master` contains a

complete list of configurable devices, cards, and pseudo-drivers.

Constructing `dfile`: Optional Second Part

The second part of *dfile* is optional, and is used to:

- Define the root device.
- Specify and set up the swap device.
- Define status and values of selected system parameters.
- Include or exclude kernel code for messages, semaphores, and shared memory features.

Lines are constructed as indicated below for each category.

1. Root device specification

`root` *devname address*

Define driver name and minor device number for root device:

devname Driver name for the device, as it appears in the alias table in `/etc/master`. For example, `cs80` for the HP 7958 disk drive.

address Minor device number (in hexadecimal without the preceeding `0x`).

2. Swap device specification

`swap auto`

Automatically configure swap on the root device to start just after the file system and extend to the end of the disk. Since there is only one root device, `auto` can be specified only once in the *dfile*, and must appear before any other swap device specification line. If `swap auto` is specified, the root device becomes the primary swap device.

`swap` *devname address swplo* `[`*nswap*`]`

Configure the swap device location and its size as specified. Arguments are interpreted as follows:

devname Driver name for the device as it appears in the alias table in `/etc/master` (for example, `cs80` for the HP 7933 404MB disk drive).

address Minor device number in hexadecimal without the preceeding `0x`; e.g., `e0100` for a disk at select code 14, HP-IB address 01.

swplo Swap area location in decimal. Boundaries are located at 1K-byte intervals.

A negative value (typically `-1`) for *swplo* specifies that a file system is expected on the device. At boot-up, the super block is read to determine the exact size of the file system, and this value is put in *swplo*. If the swap device is auto-configured, this is the mechanism used. If the super block is invalid, the entry will be skipped, so that a corrupted super block will not later cause the entire file system to be corrupted by configuring the swap area on top of it.

A positive or zero value for *swplo* specifies the minimum area that must be reserved. Zero means to reserve no area at the head of the device. *A zero value implies that there is no file system on the device.*

nswap The number (in decimal) of 1K-byte disk blocks in the swap area. Only the *nswap* parameter is optional. Zero is the default for auto-configuration.

If *nswap* is zero, the entire remainder of the device is automatically configured in as swap area.

If *nswap* is non-zero, its absolute value is treated as an upper bound for the size of the swap area. Then, if the swap area size has actually been cut back, the sign of *nswap* determines whether *swplo* remains as is, resulting in the swap area being adjacent to the reserved area, or whether *swplo* is bumped by the size of the unused area, resulting in the swap area being adjacent to the tail of the device.

3. System parameters

These parameters should not be modified without a full understanding of the ramifications of doing so (see the *HP-UX System Administrator Tasks Manual* for information about each parameter.)

Each line contains two fields. The first field can contain up to 20 characters, maximum; the second field up to 60 characters, maximum. Each line is independent, optional, and written in the following format:

parameter_name *number or formula*

System V interprocess communication consists of messages (**mesg**), semaphores (**sema**) and shared memory (**shmem**) features.

If **mesg**, **sema**, and/or **shmem** are specified as 0, the kernel code for these features is not included. If they are specified as 1, the kernel code is included; this is the default. The features can be specified independent of each other. If the code is included, the parameters listed below can be modified:

mesg	1
msgmap	*number or formula*
msgmax	*number or formula*
msgmnb	*number or formula*
msgmni	*number or formula*
msgseg	*number or formula*
msgssz	*number or formula*
msgtql	*number or formula*
sema	1
semaem	*number or formula*
semmap	*number or formula*
semmni	*number or formula*
semmns	*number or formula*
semmnu	*number or formula*
semume	*number or formula*
semvmx	*number or formula*
shmem	1
shmall	*number or formula*
smbrk	*number or formula*
shmmax	*number or formula*
shmmin	*number or formula*
shmmni	*number or formula*
shmseg	*number or formula*

FILES

/etc/master	default input master device table
/etc/master/config.sys	contains skeleton makefile
/etc/master/mkdev.sys	contains skeleton mkdev script
./conf.c	default output configuration table
./config.mk	default output *make*(1) script
./mkdev	default output *mkdev*(1M) script

SEE ALSO

make(1), mkdev(1M), master(4).

NAME

convertfs - convert a file system to allow long file names

SYNOPSIS

/etc/convertfs [*special_file*]

DESCRIPTION

convertfs converts an existing HFS file system supporting the default maximum file name length of 14 characters into one that supports file names up to 255 characters long. Once an HFS file system is converted to long file names, it cannot be restored to its original state, since the longer file names require a directory representation that is incompatible with the default HFS directory format. Since this is an irreversible operation, **convertfs** prompts for verification before it performs a conversion.

convertfs forces the system to be rebooted if the root file system is converted. When converting the root file system, the system should be in single-user mode, with all unnecessary processes terminated and all non-root file systems unmounted. Except for the root file system, **convertfs** requires that the file system to be converted be unmounted.

If invoked without any arguments, **convertfs** interactively prompts the user with a list of the file systems from **/etc/checklist**. One or more or all of the listed file systems can be selected for conversion. Typically, it is desirable to convert all of the file systems in **/etc/checklist** to avoid inconsistencies between two file systems mounted on the same system.

convertfs can also be invoked with an argument of either a block or character *special_file* of a file system to be converted. Only the block special file should be specified for a mounted root file system.

As part of the conversion process, **convertfs** performs an **fsck** on each file system (see *fsck*(1M)).

AUTHOR

convertfs was developed by HP.

SEE ALSO

mkfs(1M), newfs(1M), fs(4).

NAME

cpset - install object files in binary directories

SYNOPSIS

cpset [**-o**] *object directory* [*-mode* [*-owner* [*-group]]]

DESCRIPTION

cpset is used to install the specified *object* file in the given *directory*. The *mode*, *owner*, and *group*, of the destination file can be specified on the command line. If this data is omitted, two results are possible:

- If the user of **cpset** has administrative permissions (that is, the user's numerical ID is less than 100), the following defaults are provided:

mode	**0555**
owner	**bin**
group	**bin**

- If the user is not super-user, the default mode, owner, and group of the destination file are that of the invoker.

An optional argument of **-o** forces **cpset** to move *object* to **OLD***object* in the destination directory before installing the new object.

For example, all of the following examples have the same effect (assuming the user is an administrator), copying file **echo** into **/bin** with mode, owner, and group set to **0555 , bin , bin ,** respectively:

```
cpset echo /bin 0555 bin bin

cpset echo /bin

cpset echo /bin/echo
```

cpset utilizes file **/usr/src/destinations** to determine the final destination of a file. The locations file contains pairs of pathnames separated by spaces or tabs. The first name is the "official" destination (for example: **/bin/echo**). The second name is the new destination. For example, if **echo** is moved from **/bin** to **/usr/bin**, the entry in **/usr/src/destinations** would be:

```
/bin/echo       /usr/bin/echo
```

When the actual installation happens, **cpset** verifies that the "old" pathname does not exist. If a file exists at that location, **cpset** issues a warning and continues. This file does not exist on a distribution tape; it is used by sites to track local command movement. The procedures used to build the source are responsible for defining the "official" locations of the source.

Cross Generation

The environment variable **ROOT** is used to locate the destination file (in the form **$ROOT/usr/src/destinations**). This is necessary in the cases where cross generation is being done on a production system.

Access Control Lists (ACLs)

Use **chacl** to set optional ACL entries on a newly installed file (see *chacl*(1)).

SEE ALSO

chacl(1), make(1), install(1M), acl(5).

NAME

cron - clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in crontab files. Users can also submit their own crontab file via the *crontab*(1) command. Commands that are to be executed only once can be submitted by using the *at* command. Since *cron* never exits, it should be executed only once. This is best done by running *cron* from the initialization process through the file **/etc/rc** (see *init*(1M)).

cron only examines crontab files and *at*(1) command files during process initialization and when a file changes. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

In the HP Clustered environment, the directories **/usr/lib/cron** and **/usr/spool/cron** are context dependent files (CDFs). Each cnode is expected to run its own copy of *cron*.

Notes

On the days of daylight savings (summer) time transition (in time zones and countries where daylight savings time applies), *cron* schedules commands differently than normal.

In the following description, an *ambiguous time* refers to an hour and minute that occurs twice in the same day because of a daylight savings time transition (usually on a day during the Autumn season). A *nonexistent time* refers to an hour and minute that does not occur because of a daylight savings time transition (usually on a day during the Spring season). *DST-shift* refers to the offset that is applied to standard time to result in daylight savings time. This is normally one hour, but can be any combination of hours and minutes up to 23 hours and 59 minutes (see *tztab*(4)).

When a command is specified to run at an ambiguous time, the command is executed only once at the *first* occurrence of the ambiguous time.

When a command is specified to run a non-existent time, the command is executed after the specified time by an amount of time equal to the *DST-shift*. When such an adjustment would conflict with another time specified to run the command, the command is run only once rather than running the command twice at the same time.

For commands that are scheduled to run during all hours by specifying a * in the hour field of the crontab entry, the command is scheduled without any adjustment.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cron* behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

The following examples assume that the time zone is **MST7MDT**. In this time zone the DST transition occurs one second before 2:00 a.m. and the DST-shift is 1 hour.

Consider the following crontab entries:

# Minute	Hour	Month Day	Month	Weekday	Command
0	01	*	*	*	Job_1
0	02	*	*	*	Job_2
0	03	*	*	*	Job_3
0	04	*	*	*	Job_4
0	*	*	*	*	Job_hourly
0	2,3,4	*	*	*	Multiple_1
0	2,4	*	*	*	Multiple_2

For the period of 01:00 a.m. to 04:00 a.m. on the days of DST transition, the results will be:

Job	Times Run in Fall	Times Run in Spring
Job_1	01:00 MDT	01:00 MST
Job_2	02:00 MDT	03:00 MDT
Job_3	03:00 MST	03:00 MDT
Job_4	04:00 MST	04:00 MDT
Job_hourly	01:00 MDT	01:00 MST
	02:00 MDT	
	02:00 MST	
	03:00 MST	03:00 MDT
	04:00 MST	04:00 MDT
Multiple_1	02:00 MDT	
	03:00 MST	03:00 MDT
	04:00 MST	04:00 MDT
Multiple_2	02:00 MDT	03:00 MDT
	04:00 MST	04:00 MDT

WARNINGS

In the Spring, when there is a non-existent hour because of daylight savings time, a command that is scheduled to run multiple times during the non-existent hour will only be run once. For example, a command scheduled to run at 2:00 and 2:30 a.m. in the **MST7MDT** time zone will only run at 3:00 a.m. The command that was scheduled at 2:30 a.m. will not be run at all, instead of running at 3:30 a.m.

DIAGNOSTICS

A history of all actions taken by *cron* is recorded in **/usr/lib/cron/log**.

AUTHOR

cron was developed by AT&T and HP.

FILES

/usr/lib/cron	main *cron* directory
/usr/spool/cron	spool area
/usr/lib/cron/log	accounting information

SEE ALSO

at(1), crontab(1), sh(1), init(1M), cdf(4), queuedefs(4), tztab(4).

STANDARDS CONFORMANCE

cron: SVID2

NAME

csp - create cluster server processes

SYNOPSIS

`/etc/csp` [[`-a`|`-d`]*number*]

DESCRIPTION

The `csp` command establishes or changes the minimum number of HP Cluster server processes (CSPs) running on the system. If no arguments are specified, `csp` reads the `/etc/clusterconf` file to determine how many CSPs should be running.

Options

`csp` recognizes the following options:

`-a` *number* Add the specified *number* of CSPs to those currently running.

`-d` *number* Delete the specified *number* of CSPs from those currently running.

number (`-a` or `-d` not specified) Set the default minimum number of running CSPs to *number*. This form of the `csp` command creates the default minimum *number* of CSPs to run on a cnode.

Additional CSPs are started automatically when system load demands it, with the limitation that the number of active CSPs on a system cannot exceed the value of the HP-UX tuneable parameter `ngcsp`. `csp` is normally executed from the `/etc/rc` script.

This command can be executed only by the super-user.

AUTHOR

`csp` was developed by HP.

FILES

`/etc/clusterconf`

SEE ALSO

clusterconf(4), cluster(1M).

NAME

cstm - Command line interface to the Support Tool Manager

SYNOPSIS

/usr/diag/bin/cstm [**-m**] [**-l** *log_file*]

DESCRIPTION

cstm is a command-line user interface program for accessing a variety of support actions provided by the Support Tool Manager. **cstm** is a complementary interface to **xstm** (a graphical interface for the X11 environment) and MSTM (a menu-based interface for any HP 2392 terminal (or compatible) or on any X terminal). **cstm** can be run on an HP compatible ASCII terminal or on any X terminal.

When **cstm** is started, a banner is displayed; followed by a prompt. The "map" command may be used to generate a map of the I/O and system configuration which includes processors, I/O cards, and peripherals. The map includes the physical path of each device, a brief description of each device, and the status of all support operations that have been initiated on each device.

Note that when **cstm** is run for the first time, executing the initialization phase takes longer than when it is run again. In this sense, the initial execution of **cstm** is the equivalent of using the **-m** option (discussed below).

A **help** command is provided, which displays either a brief list of commands provided by **cstm**, or more detailed information on a particular command. To obtain the command list, type **help** or **?**. Help information for a particular command can be obtained by following the **help** command with the name of the command for which help is desired.

Operations that can be performed on a specific device depend on the support tool functions available for the type of device. Typically these actions are **verify** and **diagnose** or **exercise**. **verify** operations consist of tasks that access the device in a way that a typical user would, to determine if the device is functional. **diagnose** operations perform hardware diagnostic tests on the selected device. **exercise** operations attempt to stress the device.

Support operations for a given device are initiated by typing the appropriate command followed by the physical path of the device as shown on the output from the **map** command. For example, **verify 2/0/1** results in a verifier being run on device **2/0/1**. Many commands accept parameters such as **loop**, **time**, **all**, etc. The exact syntax, including allowable parameters for a given command, can be determined using the **help** command.

When an operation such as **diagnose**, **exercise**, or **verify** is selected, a message is displayed indicating that the operation was initiated. If a **map** command is issued prior to the completion of the operation, the corresponding status for the selected device is shown as **Running**, indicating that a test is in progress. When the action has completed, a message is displayed indicating the result of the operation. Additionally, the status in the map is changed to reflect the result. Typical result statuses are **Success**, **Failed**, and **Warning**. In the case of **Warning** and **Failed**, the result is displayed in inverse video for easier recognition. If an operation fails, detailed information concerning the exact nature of the failure can be obtained by issuing the **viewlog** command for the device. This action creates a **vi** editing session on the failure-log file.

Some actions may require assistance from the user, such as mounting a tape and making sure that the tape drive is on line. When such operator intervention is required, a prompt is displayed to which the user must enter a response.

Options

cstm recognizes the following options and command-line arguments:

-m At start-up, force a search of the physical devices on the system and of the diagnostic programs supported. When this option is used, **cstm** takes longer to initialize to the point where user interaction begins. This option is required when system configuration is changed or when a new diagnostic program is installed using **sysdiag**.

-l *log_file* Specifies the name of the file to which log events that occur during the time the application is active are posted. The default is **./stm.log**. To review the contents of this log, use the **viewlog** command with no parameters.

WARNINGS

Due to the nature of this application and its associated processes, overall system performance may be degraded while CSTE is running.

AUTHOR

`cstm` was developed by HP.

FILES

Session Log File

The session log (default name `stm.log`) contains a detailed history of the actions performed by `cstm`. This log begins with a chart indicating the system and I/O configuration for the system and records the results for each action performed during the session. Each line of the chart specifies the location of the device, a description of the device, the current status of the diagnose action, and the current status of the verify action.

Note that the `Diagnostic`, `Verifier`, and `Exerciser` status columns will contain entries of `N/A`, `Untested`, or `Incompl.`. `N/A` signifies that the corresponding action is not available for the device. `Untested` signifies that the action has not been invoked; always the case when `cstm` is first invoked. `Incompl.` signifies that an action was attempted, but was unable to complete.

Other Files

`/usr/diag/bin/am`	support application manager
`/usr/diag/bin/dtm`	diagnostic tool manager
`/usr/diag/bin/DTMDUI.sh`	diagnostic interface shell
`/usr/diag/bin/cpudaf`	cpu access function
`/usr/diag/bin/diskdaf`	disk device function
`/usr/diag/bin/fpudaf`	floating point access function
`/usr/diag/bin/graphicsdaf`	graphics device access verifier
`/usr/diag/bin/idiskdaf`	removable-media disk device access function
`/usr/diag/bin/memdaf`	memory access function
`/usr/diag/bin/tapedaf`	tape device access function
`/usr/diag/bin/CSVER000`	NLS message catalog for the platform
`/usr/diag/bin/CCSTM000`	NLS message catalog for CSTM

SEE ALSO

mstm(1M), sysdiag(1M), xstm(1M).

(Series 800 Only)

NAME

cuegetty - set terminal type, modes, speed, and line discipline for *cue*(1)

SYNOPSIS

/bin/cuegetty [**-L** *nls_language*] [**-h**] [**-t** *timeout*] *line* [*speed*]

DESCRIPTION

cuegetty, which is very similar to *getty*(1M), is the second process in the series, (*init-cuegetty-cue-work*session) that ultimately connects a user with the HP-UX CUE system. It is invoked by **init** to monitor the terminal lines configured on a system (see *init*(1M)). Each **cuegetty** process resets its process group using **setpgrp**, opens a particular terminal line, and usually sleeps in the **open()** until the machine senses a hardware connection for the terminal. When **open()** returns, **cuegetty** attempts to adapt the system to the terminal speed and type, and displays the contents of the **/etc/issue** file, if it exists. Lastly, **cuegetty** invokes **cue** which displays the Login screen and performs user validation (see *cue*(1)).

To start **cuegetty**, an entry for **cuegetty** should be placed in the **/etc/inittab** file. A typical CUE entry in the **/etc/inittab** file resembles the following:

```
cue:2:respawn:/bin/cuegetty -L french -h tty0p1
```

See **/bin/cue.etc/cue.inittab** for an example **/etc/inittab** file. See *cue*(1) for more details on the CUE system.

Configuration Options and Arguments

cuegetty recognizes the following arguments:

line Name of a tty line in **/dev** to which **cuegetty** is to attach itself. **cuegetty** uses this string as the name of a file in the **/dev** directory to open for reading and writing. By default **cuegetty** forces a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. However, when **cuegetty** is run on a direct port, **cuegetty** does not force a hangup on the line since the driver ignores changes to zero speed on ports open in direct mode (see *modem*(7)).

-L **nls_language** is used to set the language for the CUE login screens. If the message catalog, **cue.cat**, does not exist for **nls_language**, the default native language, C, is used.

-h Tells **cuegetty** not to force a hangup on the line before setting the speed to the default or specified speed.

-t *timeout* **cuegetty** exits if the open on the line succeeds and no one types anything within *timeout* seconds.

speed A label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells **cuegetty** at what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by typing a *break* character). The default *speed* is 300 baud.

When no optional arguments appear on the command line, **cuegetty** sets the terminal interface as follows:

- Interface *speed*: 300 baud,
- Raw mode (awaken on every character),
- Echo suppressed,
- Parity: either,
- New-line characters: convert to carriage-return, line-feed pair,
- Expand tabs on the standard output.
- Type login message then read user's name, one character at a time.
- If a null character (or framing error) is received, assumed it to be the result of the user pushing the "break" key. This causes *cuegetty* to attempt the next *speed* in the series. The series that *cuegetty* tries is determined by what it finds in **/etc/gettydefs**.

(Series 800 Only)

After interface set-up is complete, *cue* is started to accept and validate the user name and password.

DEPENDENCIES

cuegetty is available only on Series 800 systems, and is compatible only with the following terminals:

HP700/92 HP700/94 HP2392 HP2394

FILES

/etc/gettydefs	contains speed and terminal settings used by **cuegetty**
/etc/inittab	**init** reads this file to determine which processes to spawn
/etc/issue	contains issue identification data
/bin/cue.etc/cue.inittab	sample **inittab** file with **cuegetty** entry

SEE ALSO

cue(1), env(1), environ(5), hpnls(5), init(1M), ioctl(2), getty(1M), gettydefs(4), inittab(4), lang(5), nlsinfo(1), termio(7).

NAME

dcopy - copy file system with compaction.

SYNOPSIS

dcopy -v | **-f** *fsize* [**:** *isize*] | **-s***cyl* **:** *skip* | **-d** *source_fs destination_fs*

DESCRIPTION

dcopy copies an existing file system (*source_fs*) to a new file system (*destination_fs*), appropriately sized to hold the reorganized results. For best results, source file system should be a raw device, and the destination file system should be a block device. Always run **dcopy** on unmounted file systems (in the case of the root file system, copy to a new minidisk).

If no options are specified, **dcopy** copies files from *source_fs*, compressing directories by removing vacant entries and spacing consecutive blocks in a file by the optimal rotational gap. If options such as **-f** or **-s** are specified, the destination file system structure will be different from that of the source file system.

dcopy makes the destination file system identical to the source file system and preserves the pack and volume labels. Thus, to compress a file system without moving it, use **dcopy** to copy the files to another file system and **dd** to copy the file back (see *dd*(1)).

Directory compression is accomplished by running **dcopy** on the primary copy of the file system and allowing the modified directories to propagate to the other copies of the file system in the normal manner.

Options

dcopy recognizes the following options and command-line arguments:

-f*fsize*[**:***isize*] Specifies the file system and inode-list size in blocks. If this option is not specified, the source file-system value is used.

-s*cyl* **:***skip* Supply device information for creating the best organization of blocks in a file. *cyl* is the number of block per cylinder; *skip* is the number of blocks to skip.

-v Report size of source and destination file system.

-d *source_fs* Move subdirectories to the beginning of directories.

EXAMPLES

dcopy can be executed with or without options. If no options are specified as in this example, source and destination file systems are identical. Any differences between the two file systems lie only in the available disk space.

```
dcopy /dev/rdsk/c2d0s4 /dev/dsk/c2d0s5
```

If options are specified, expect a major difference between the source and destination file system structure:

```
dcopy -f40960:260 -s45:5 -d /dev/rdsk/c2d0s4 /dev/dsk/c2d0s5
```

WARNINGS

dcopy produces invalid results if run on a mounted file system. **dcopy** Figures specified in option arguments cannot be smaller than corresponding figures in the source file system.

AUTHOR

dcopy was developed by HP.

NAME

devnm - device name

SYNOPSIS

/etc/devnm [*name ...*]

DESCRIPTION

For each *name* specified, **devnm** identifies the special file associated with the mounted file system where the named file or directory resides. The full path name must be given.

EXAMPLES

The command:

```
/etc/devnm /usr
```

produces

```
/dev/dsk/c1d0s9 /usr
```

if **/usr** is mounted on **/dev/dsk/c1d0s9**.

In the HP Clustered environment, pathnames are displayed with context-dependent files expanded.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

brc(1M).

STANDARDS CONFORMANCE

devnm: SVID2

NAME

df - report number of free disk blocks

SYNOPSIS

df [**-t**][**-f**][**-b**][**-l**|**-L**][*file_systems*]

DESCRIPTION

df prints the number of free 512-byte blocks and free inodes available for on-line file systems by examining the counts kept in the super-block or super-blocks. *file_systems* can be specified either by device name (e.g., **/dev/dsk/0s1**) or by mounted directory name (e.g., **/usr**). If *file_systems* is not specified, the free space on all of the mounted file systems is printed.

Options

df recognizes the following options:

-t Total allocated block figures are also reported.

-f Only an actual count of the blocks in the free list is made (free inodes are not reported). When this option is specifed, **df** reports on raw devices.

-b Report total number of blocks allocated for swapping to the file system as well as the number of blocks free for swapping to the file system.

-l In the HP Clustered environment, display information for only HFS and CDFS file systems mounted on the local cnode (NFS mounts are not displayed).

-L In the HP Clustered environment, display information for file systems that can be unmounted from the local cnode. (Includes file systems mounted on the local node and cluster-wide NFS mounts).

When **df** is used on an HFS file system, the file space reported is the space available to the ordinary user, and does not include the reserved file space specified by **fs_minfree**. Unreported reserved blocks are available only to users who have appropriate privileges. See *fs*(4) for information about **fs_minfree**.

When **df** is used on NFS file systems, the number of inodes is displayed as –1 . This is due to to super-user access restrictions over NFS.

WARNINGS

df does not account for any disk space reserved for swap space, or space used for the HFS boot block (8k bytes, 1 per file system), HFS superblocks (8k bytes each, 1 per disk cylinder), HFS cylinder group blocks (1k-8k bytes each, 1 per cylinder group), and inodes (currently 128 bytes reserved for each inode). Non-HFS file systems may have other items that this command does not account for.

In the HP Clustered Environment, device and mountpoint pathnames are displayed with context-dependent files expanded.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

du(1), fsck(1M), fs(4), mnttab(4).

STANDARDS CONFORMANCE

df: SVID2, XPG2, XPG3

NAME

DIAGINIT, DIAGMON, DEMLOG, MEMLOGP - Online diagnostic system

SYNOPSIS

`/usr/diag/bin/DIAGINIT`

diagnostic daemon `DIAGMON`

diagnostic daemon `DEMLOG`

diagnostic daemon `MEMLOGP`

DESCRIPTION

The online diagnostic subsystem is used to diagnose suspected system hardware problems. The diagnostic subsystem uses a monitor (`DIAGMON`) to manage the system resources needed to test a diagnosable piece of hardware. The diagnostic subsystem also logs selected errors for use by field personnel.

`DIAGINIT` Program sets up system resources needed by the diagnostic subsystem and launches the diagnostic monitor daemon, `DIAGMON`.

`DIAGMON` Daemon controls the system resources needed to diagnose a piece of system hardware. Also launches the remaining diagnostic daemons, and logs diagnostic events in the system log files located in `/usr/adm/diag`.

`DEMLOG` This daemon reads diagnostic events from the kernel via `diag0` (see *diag0*(7), and sends them to `DIAGMON` which logs them.

`MEMLOGP` Daemon periodically polls memory for errors, and logs them to the `MEMLOGP` log file in `/usr/adm/diag`.

WARNINGS

The online diagnostic subsystem uses System V inter-process communication (IPC) resources. If an application is run on the system that exhausts these resources, the online diagnostic subsystem may be adversely affected. It is recommended that the system defaults in `/etc/master` for System V IPC not be reduced.

Series 800

If `DEMLOG` does not read diagnostic events from `diag0` as fast as they are logged by the kernel, a warning message, `Warning: DIAG0 -- message queue full`, appears on the console.

DEPENDENCIES

Series 700

`DEMLOG` does not exist on Series 700 systems.

AUTHOR

The online diagnostic subsystem was developed by HP.

FILES

`/hp-ux`	
`/dev/diag0`	Series 800 pseudo-driver device file
`/dev/diag1`	Series 700 pseudo-driver device file
`/dev/dmem`	memory device file
`/dev/diag`	directory containing diagnostic device files
`/usr/adm/diag`	directory containing diagnostic log files
`/usr/diag/bin/DIAGINIT`	
`/usr/diag/bin/DIAGMON`	
`/usr/diag/bin/DEMLOG`	
`/usr/diag/bin/MEMLOGP`	

SEE ALSO

sysdiag(1M), diag0(7), diag1(7).

Online diagnostic subsystem manuals.

NAME

diskinfo - describe characteristics of a disk device

SYNOPSIS

`/etc/diskinfo` [`-b` | `-v`] *character_devicefile*

DESCRIPTION

`diskinfo` determines whether the character special file named by *character_devicefile* is associated with a SCSI, CS/80, or Subset/80 disk drive; if so, `diskinfo` summarizes the disk's characteristics.

Options

`diskinfo` recognizes the following options:

`-b` Return the size of the disk in 1024-byte sectors.

`-v` Display a verbose summary of all the information available from the device (since the information returned by CS/80 drives and SCSI drives differs, the associated descriptions differ also).

CS/80 devices return the following:

device name
number of bytes/sector
geometry information
interleave
type of device
timing information

SCSI disk devices return the following:

vendor and product ID
device type
size (in bytes and in logical blocks)
bytes per sector
revision level
SCSI conformance level data

`diskinfo` displays information about the following characteristics of disk drives:

Vendor name	Manufacturer of the drive (SCSI only)
Product ID	Product identification number or ASCII name
Type	CS/80 or SCSI classification for the device
Disk	Size of disk specified in bytes
Sector	Specified as bytes per sector

Both size of disk and bytes per sector represent formatted media.

DEPENDENCIES

General

`diskinfo` supports only CS/80, subset/80, and HP SCSI disk devices.

SCSI

The SCSI specification provides for a wide variety of device-dependent formats. For non-HP devices, `diskinfo` might be unable to interpret all the data returned by the device. Refer to the drive operating manual accompanying the unit for more information.

AUTHOR

`diskinfo` was developed by HP.

SEE ALSO

lsdev(1M), mkdev(1M), disktab(4), disk(7).

NAME

disksecn - calculate default disk section sizes

SYNOPSIS

disksecn [**-p** | **-d**] [**-b** *block_size*] [**-n** *disk_name*]

DESCRIPTION

disksecn is used to calculate the disk section sizes based on the Berkeley disk partitioning method.

disksecn recognizes the following options:

-p Produce tables suitable for inclusion in the device driver.

-d Produce tables suitable for generating the disk description file **/etc/disktab**.

-b *block_size* When generating the above tables, use a sector size of *block_size* bytes, where *block_size* can be **256**, **512**, **1024**, or **2048**. Defaults to DEV_BSIZE (defined in **<sys/param.h>**) if not specified.

-n *disk_name* Specifies the disk name to be used in calculating sector sizes; for example, **hp7912** or **hp7945**. If an unknown disk name is specified, *disksecn* prompts the user for the necessary disk information.

If neither **-p** nor **-d** table selection switches are specified a default table of the section sizes and range of cylinders used is output.

Disk section sizes are based on the total amount of space on the disk as given in the table below (all values are supplied in units of 256-byte sectors). If the disk is smaller than approximately 44 Mbytes, *disksecn* aborts and returns the message **disk too small, calculate by hand**.

Section	44-56MB	57-106MB	107-332MB	333+MB
0	97120	97120	97120	97120
1	39064	39064	143808	194240
3	39064	39064	78128	117192
4	unused	48560	110096	429704
6	7992	7992	7992	7992
10	unused	unused	unused	516096

NOTE:
It is important to note the difference between the block size passed into *disksecn* via the **-b** switch argument and the sector size the user is asked to input when an unknown disk name is passed to *disksecn* via the **-n** switch argument.

The block size is the sector size that *disksecn* assumes the disk to have when it prints the requested tables. All information printed in the tables is adjusted to reflect this assumed sector size (block size) passed in by the user. The sector size requested by *disksecn* when an unknown disk name is passed does not necessarily have to be the same as the assumed sector size (block size) passed in by the **-b** switch argument.

For example, a user wants to see the device driver tables for the disk named **hp7945** with an assumed sector size (block size) of 256 bytes. The user has the following information about the **hp7945** disk:

Disk type = winchester
Sector size = 512
Number of sectors per track (512 byte sectors) = 16
Number of tracks = 7
Number of cylinders = 968
Revolutions per minute = 3600

The user invokes *disksecn* by typing the following command:

disksecn -p -b 256 -n hp7945

Assuming that **hp7945** is an unknown disk name, *disksecn* prompts the user for the necessary disk information. The user should input the information as shown above, reflecting a sector size of 512 bytes. All the information will be adjusted within *disksecn* to reflect the assumed sector size (block size) of 256 bytes, passed as the argument of the **-b** switch, before the requested device driver table is output.

This adjustment also takes place when the disk name is known and an assumed sector size (block size) is passed in as the argument of the **-b** switch which is not DEV_BSIZE bytes, the assumed sector size (block size) used to create the **etc/disktab** file.

RETURN VALUE

disksecn returns the following values:

0 Successful completion.
1 Usage error.
2 User did not input parameters for an unknown disk.
3 Disk too small or an invalid block size.

disksecn aborts and prints an error message under the following conditions:

- *disksecn* was invoked without specifying a disk name.
- Requested both **-p** and **-d** switch.
- Illegal block size requested.
- Unknown disk name was specified and user did not supply disk information.
- Disk's maximum storage space is less than approximately 44 MB.

WARNINGS

Alternate names are not included in the output when the **-d** switch is used.

Blanks are required in the command line between each of the switches when invoking *disksecn*.

A blank is required between the **-n** switch and the disk name argument to that switch. For example:

disksecn -p -b 1024 -n hp9712

disksecn does not save the block size used to generate the **/etc/disktab** disk description file. The system assumes that the block size used was DEV_BSIZE when it reads the information stored in the **etc/disktab** file.

AUTHOR

disksecn was developed by the University of California, Berkeley.

FILES

/etc/disktab

SEE ALSO

disktab(4).

NAME

diskusg - generate disk accounting data by user ID

SYNOPSIS

diskusg [*options*] [*files*]

DESCRIPTION

diskusg generates intermediate disk accounting information from data in *files*, or the standard input if omitted. **diskusg** outputs lines on the standard output, one per user, in the following format:

uid login #blocks

where:

uid User's numerical user ID,

login User's login name, and

#blocks Total number of disk blocks allocated to this user.

diskusg normally reads only the inodes of file systems for disk accounting. In this case, *files* are the special filenames of these devices.

Options

diskusg recognizes the following options:

-s Input data is already in **diskusg** output format. **diskusg** combines all lines for a single user into a single line.

-v verbose. Print a list on standard error of all files that are charged to no one.

-i *fnmlist* Ignore the data on those file systems whose file system name is in *fnmlist*. *fnmlist* is a list of file system names, separated by commas or enclosed within quotes. **diskusg** compares each name in this list with the file system name stored in the volume ID if it exists.

-p *file* Use *file* as the name of the password file to generate login names. **/etc/passwd** is used by default.

-u *file* Write records to *file* of files that are charged to no one. Records consist of the special file name, the inode number, and the user ID.

The output of **diskusg** is normally the input to **acctdisk** (see *acct*(1M)) which generates total accounting records that can be merged with other accounting records. **diskusg** is normally run in **dodisk** (see *acctsh*(1M)).

EXAMPLES

The following generates daily disk accounting information:

```
for i in /dev/rp00 /dev/rp01 /dev/rp10 /dev/rp11; do
    diskusg $i > dtmp.'basename $i' &
done
wait
diskusg -s dtmp.* | sort +0n +1 | acctdisk > disktacct
```

FILES

/etc/passwd used for user-ID-to-login-name conversions

SEE ALSO

acct(1M), acctsh(1M), volcopy(1M), acct(4).

STANDARDS CONFORMANCE

diskusg: SVID2

NAME

dmesg - collect system diagnostic messages to form error log

SYNOPSIS

`/etc/dmesg [-]`

DESCRIPTION

`dmesg` looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed by the system when unusual events occur (such as when system tables overflow or the system crashes). If the - argument is specified, `dmesg` computes (incrementally) the new messages since the last time it was run and places these on the standard output. This is typically used with `cron` (see *cron*(1)) to produce the error log `/usr/adm/messages` by running the command:

```
/etc/dmesg - >> /usr/adm/messages
```

every 10 minutes.

WARNINGS

The system error message buffer is of small, finite size. `dmesg` is run only every few minutes, so there is no guarantee that all error messages will be logged.

AUTHOR

`dmesg` was developed by the University of California, Berkeley.

FILES

`/usr/adm/messages`
: error log (conventional location)

`/usr/adm/msgbuf`
: memory scratch file for - option

NAME

dpp - dedicated ports parser, used by DDFA software

SYNOPSIS

dpp *dp_file* [**-l** *log_file*] [**-c**] [**-k**] [**-p** *ocd_program*]

DESCRIPTION

This file is part of the HP Datacommunications and Terminal Controller (DTC) Device File Access (DDFA) software. It parses the **Dedicated Ports file** (**dp**) and calls the **Outbound Connection Daemon** (**ocd**) to spawn a daemon for each valid entry in **dp**.

dpp can be run from the shell or it can be included in **netlinkrc** to automatically run the DDFA software each time the system is booted. Only user **root** can run **dpp**.

Refer to *ddfa*(7) for information about how to configure and install DDFA software.

Options

dpp recognizes the following options and command-line arguments:

dp_file — Mandatory argument, and must be the first field. *dp_file* must be executable and meet the specifications given in *dp*(4). The **dp** file defines the link between the DTC port and the device file used by applications to access the port. If *dp_file* is modified, **dpp** must be run again to activate the changes.

-l *log_file* — Log messages to *log_file*. If no log file is specified, all error messages are logged to the screen.

If this option is used, the *log_file* argument is required. If the specified file does not already exist, it is created. *log_file* must be non-executable, and readable by **dpp**.

-p *ocd_program* — The default path for **ocd** is **/etc/ocd**; if the path is different, it must be specified using the **-p** option. The **ocd** must be executable with access rights for **dpp**.

dpp starts a process for each valid entry in the **dp** file and starts execution of **ocd**.

-k — Remove the device file corresponding to each valid entry in the **dp** file, then launch **ocd** for each valid entry.

ocd normally creates and removes devices files. However, if the process is killed badly, such as with **kill -9**, the device file may remain. If the system is rebooted, the **-k** option can be specified to restart all **dp** file entries correctly. Deleting the device file eventually causes the **ocd** process (if any) to exit.

If a corresponding **ocd** no longer exists, the device file is removed by any following invocation of an **ocd** that requires the same device file.

-c — Parse the **dp** file and log all bad entries. This option is useful for debugging the **dp** file before running it properly. The **-p***ocd_program* entry is ignored if **-c** is used.

The **netlinkrc** script can be modified to include **dpp**, as in:

```
/etc/dpp /etc/ddfa/dp_file -k
```

dpp runs, scans all the entries in the **dp** file, creates a process for each valid entry then exits. It is recommended that the **-k** option be used, and that a log file be specified.

If the **dp** file is modified, **dpp** must be run again to detect the changes.

KILLING DAEMONS

Outbound-connection daemons (**ocd**s) should be killed using **kill -15**. Do not use **kill -9** because it does not remove the device file. **ocd** verifies the validity of an existing pseudonym before trying to use it. **dpp** and **ocd** use data stored in file **/etc/utmp.dfa** to verify whether a process still owns a pseudonym before taking it over. If **ocd** finds an unowned pseudonym, it uses it.

ERRORS

Error messages are logged for bad arguments, bad file entries and `ocd` process-creation errors. By default, they are logged to the standard output. If the `-l`*error_log* option is used, they are appended to the log file

Error messages are generally self-explanatory, but a brief explanation of some has been included in the following list.

`error 0:` *dp_file* is mandatory.

`error 1:` `dp` file must be the first argument.

`error 2:` Cannot read `dp` file.

The `dp` file either does not exist or cannot be accessed with current access privileges.

`error 3:` No log file defined (`-l` option).

`error 4:` Cannot create log file (`-l` option).

The log file cannot be created, either because of an invalid path or because of insufficient access privileges.

`error 5:` Cannot access log file (`-l` option).

The log file cannot be accessed, either because of an invalid path or because of insufficient access privileges. The log file must be readable by anyone.

`error 6:` No `ocd` file defined in program option.

`error 7:` Cannot execute `ocd` program (`-p` option).

The `ocd` program specified in the `-p` option either does not exist, or is not an executable file with current access privileges.

`error 8:` Cannot purge device file (`/dev/xxx`).

The `-k` option has been specified, the device file exists, but it cannot be purged because of insufficient access privileges.

`error 9:` Cannot execute default program (`/etc/ocd`).

The default `ocd` cannot be executed, either because of insufficient access privileges or because it has not been correctly installed.

`error 10:` Entry ignored (Bad IP address).

The `dp` file entry specified does not have a valid IP address.

`error 11:` Entry ignored (no *board*/*port* info).

`error 12:` Entry ignored (Bad port number).

The port specified is either not a decimal value or a string composed of `x` or `X` characters.

`error 13:` Entry ignored (Bad board number).

The board specified is either not a decimal value or is not a string composed of `x` or `X` characters.

`error 14:` No more processes available on system.

The `ocd` program specified cannot be started because there are no processes available on the system.

`error 15:` Entry ignored (no device name).

`error 16:` Entry ignored (Bad device name).

The device file specified cannot be created, either because of an invalid path or insufficient access rights.

`error 17:` Entry ignored (Bad *config_name*).

The specified config file cannot be read, either because of an invalid path or insufficient access rights.

FILES

`/etc/dpp`

```
/etc/ocdebug
/etc/ocd
/etc/dpp_login.bin
/etc/utmp.dfa
/etc/newconfig/ddfa/pcf
/etc/newconfig/ddfa/dp
```

SEE ALSO

ocd(1M), ocdebug(1M), dp(4), pcf(4) ddfa(7).

NAME

drm_admin - Data Replication Manager administrative tool

SYNOPSIS

/etc/ncs/drm_admin [*-version*]

DESCRIPTION

drm_admin administers servers based on the Data Replication Manager (DRM) such as *glbd*, the replicated version of the Global Location Broker (GLB).

drm_admin is used to inspect or modify replica lists, merge databases to force convergence among replicas, stop servers, and delete replicas.

The role of *drm_admin* is to administer the replication of databases, not to change the data they contain. For example, use *drm_admin* to merge two replicas of the GLB database; use *lb_admin* to add a new entry to the database. Also, although *drm_admin* can stop or delete a GLB replica, you must invoke **glbd** directly if you want to start or create a replica.

Once invoked, *drm_admin* enters an interactive mode, in which it accepts the commands described below.

OPTIONS

-version

Display the version of NCK that this *drm_admin* belongs to, but do not start the tool.

COMMANDS

Most *drm_admin* commands operate on a default object (*default_obj*) at a default host (*default_host*). Together, *default_obj* and *default_host* specify a default replica. Defaults are established by the **set** command and are remembered until changed by another **set**.

Currently, the only known object is **glb**.

Some *drm_admin* commands operate on a host other than the default. We identify this host as *other_host*.

The host name you supply as a *default_host* or an *other_host* takes the form *family:host*, where the host can be specified either by its name or by its network address. For example, **ip:bertie**, **ip:#192.5.5.5**, **dds://jeeves**, and **dds:#101a.57f95** are acceptable host names. (All HP-UX hosts have **ip** as the *family*.)

addrep *other_host*

Add *other_host* to the replica list at *default_host*. The replica at *default_host* will propagate *other_host* to all other replica lists for *default_obj*.

chrep -from *other_host* **-to** *new_other_host*

Change the network address for *other_host* in the replica list at *default_host* to *new_other_host*. The replica at *default_host* will propagate this change to all other replica lists for *default_obj*. The **chrep** command fails if a replica of *default_obj* is running at *other_host* or if *other_host* is not on the replica list at *default_host*.

delrep *other_host*

Delete the replica of *default_obj* at *other_host*. The **delrep** command tells the replica at *other_host*

1. To propagate all of the entries in its propagation queue.
2. To propagate a delete request to all other replicas, causing *other_host* to be deleted from all other replica lists for *default_obj*.
3. To delete its copy of *default_obj*.
4. To stop running.

The **delrep** command returns you immediately to the *drm_admin* prompt, but the actual deletion of the replica can take a long time in configurations that are not stable and intact. To check whether the daemon for the deleted replica has stopped, list the processes running on its host.

info

Get status information about the replica for *default_obj* at *default_host*.

lrep [**-d**] [**-clocks**] [**-na**]

List replicas for *default_obj* as stored in the replica list at *default_host*.

The **-d** option lists deleted as well as existing replicas.

The **-clocks** option shows the current time on each host and indicates clock skew among the replicas.

The **-na** option lists the network address of each host.

merge { **-from** | **-to** } *other_host*
The **merge** command copies entries in the *default_obj* database and replica list from one replica to another. It copies an entry if no corresponding entry exists in the destination database or if the corresponding entry in the destination database bears an earlier timestamp.

A merge does not cause entries to be propagated. The database and replica list at the origination are not changed.

The **-from** option copies entries from the *default_obj* database and replica list at *other_host* to the *default_obj* database and replica list at *default_host*.

The **-to** option copies entries from the database and replica list at *default_host* to the database and replica list at *other_host*.

A **merge -from** followed by a **merge -to** causes the replicas at the two hosts to converge.

merge_all
The **merge_all** command uses *default_host* as the hub for a global merge of all replicas for *default_obj*. For each host on the replica list at *default_host*, a **merge_all** first does a **merge -from**, then does a **merge -to**. All replicas of *default_obj* are thereby forced into a consistent state. The **merge_all** operation does not cause any entries to be propagated.

Perform a **merge_all** when:

- A replica is purged,
- A replica is reset,
- A replica has been incommunicado for two weeks or more,
- A replica dies (for example, its database is destroyed by a disk failure).

monitor [**-r** *n*]
This command causes *drm_admin* to read the clock of each replica of *default_obj* every *n* minutes and report any clock skews or non-answering replicas. If **-r** is not specified, the period is 15 minutes.

purgerep *other_host*
The **purgerep** command purges *other_host* from the replica list at *default_host*. The replica at *default_host* then propagates a delete request to the replicas at the hosts remaining on its list, thereby removing *other_host* from all other replica lists for *default_obj*. The delete request is not sent to *other_host*.

A **purgerep** can cause data to be lost and should only be used when a replica has died. A **merge_all** operation after the **purgerep** is strongly recommended to prevent the remaining replicas of the *default_obj* database from becoming inconsistent. If the purged replica is still running, it should be **reset**.

Recommendation: Use **chrep** (rather than **addrep** and **purgerep**) to change entries on the replica list.

quit
Quit the *drm_admin* session.

reset *other_host*
Reset the replica of *default_obj* at *other_host*.

The **reset** command tells the replica at *other_host* to delete its copy of *default_obj* and stop running. It does not cause *other_host* to be deleted from any other replica lists. This command can cause data to be lost unless a successful **merge_all** is done first.

set [**-o** *obj_name*] **-h** *host_name*
Set the default object and host. All subsequent commands operate on *obj_name*. Subsequent commands that do not specify a host are sent to *host_name*. If you do not specify the **-o** option, *drm_admin* keeps the current *default_obj*.

If you use **set** with the **-o** option, *drm_admin* checks the clocks at all hosts with replicas of the specified object.

stop
Stop the server for *default_obj* that is running at *default_host*.

EXAMPLES
Start *drm_admin*, set the default object to **glb**, and set the default host to **mars**:

```
$ /etc/ncs/drm_admin
drm_admin: set -o glb -h ip:mars
        Default object: glb  default host: ip:mars  state: in service
        Checking clocks of glb replicas
        ip:mars                                  1987/04/09.17:09
        ip:pluto                                 1987/04/09.17:09
        ip:mercury                               1987/04/09.17:07
```

SEE ALSO
glbd(1M), lb_admin(1M).

NAME

dump, rdump - incremental file system dump, local or across network

SYNOPSIS

/etc/dump [*option* [*argument* ...] *filesystem*]
/etc/rdump [*option* [*argument* ...] *filesystem*]

DESCRIPTION

dump and **rdump** copy to magnetic tape all files in the *filesystem* that have been changed after a certain date. This information is derived from the files **/etc/dumpdates** and **/etc/checklist**. *option* specifies the date and other options about the dump. *option* consists of characters from the set **0123456789bdfnsuWw**.

Options

0-9 This number is the "dump level". All files modified since the last date stored in file **/etc/dumpdates** for the same filesystem at lesser levels will be dumped. If no date is determined by the level, the beginning of time is assumed. Thus, the option **0** causes the entire filesystem to be dumped.

b The blocking factor is taken from the next argument (default is 10 if not specified). Block size is defined as the logical record size times the blocking factor. **dump** writes logical records of 1024 bytes. When dumping to tapes with densities of 6250 BPI or greater without using the **b** option, the default blocking factor is 32.

d The density of the tape (expressed in BPI) is taken from the next *argument*. This is used in calculating the amount of tape used per reel. The default is 1600.

f Place the dump on the next *argument* file instead of the tape. If the name of the file is **-**, **dump** writes to the standard output. When using **rdump**, this option should be specified, and the next argument supplied should be of the form *machine*:*device*.

n Whenever **dump** and **rdump** require operator attention, notify all users in group **operator** by means similar to that described by *wall*(1).

s The size of the dump tape is specified in feet. The number of feet is taken from the next *argument*. When the specified size is reached, **dump** and **rdump** wait for reels to be changed. The default tape size is 2300 feet.

u If the dump completes successfully, write on file **/etc/dumpdates** the date when the dump started. This file records a separate date for each filesystem and each dump level. The format of **/etc/dumpdates** is user-readable and consists of one free-format record per line: filesystem name, increment level and dump date in *ctime*(3C) format. File **/etc/dumpdates** can be edited to change any of the fields if necessary.

W For each file system in **/etc/dumpdates**, print the most recent dump date and level, indicating which file systems should be dumped. If the **W** option is set, all other options are ignored and **dump** exits immediately.

w Operates like **W**, but prints only filesystems that need to be dumped.

If no arguments are given, *option* is assumed to be **9u** and a default file system is dumped to the default tape.

Sizes are based on 1600-BPI blocked tape; the raw magnetic tape device must be used to approach these densities. Up to 32 read errors on the filesystem are ignored. Each reel requires a new process; thus parent processes for reels already written remain until the entire tape is written.

rdump creates a server, **/etc/rmt**, on the remote machine to access the tape device.

dump and **rdump** require operator intervention for any of the following conditions:

- end of tape,
- end of dump,
- tape-write error,
- tape-open error, or
- disk-read error (if errors exceed threshold of 32).

In addition to alerting all operators implied by the **n** option, **dump** and **rdump** interact with the control terminal operator by posing questions requiring **yes** or **no** answers when it can no longer proceed or if something is grossly wrong.

Since making a full dump involves considerable time and effort, **dump** and **rdump** each establish a checkpoint at the start of each tape volume. If, for any reason, writing that volume fails, **dump** and **rdump** will, with operator permission, restart from the checkpoint after the old tape has been rewound and removed and a new tape has been mounted.

dump and **rdump** periodically report information to the operator, including typically low estimates of the number of blocks to write, the number of tapes it will require, time needed for completion, and the time remaining until tape change. The output is verbose to inform other users that the terminal controlling **dump** and **rdump** is busy and will be for some time.

Access Control Lists (ACLs)

The optional entries of a file's access control list (ACL) are not backed up with **dump** and **rdump**. Instead, the file's permission bits are backed up and any information contained in its optional ACL entries is lost (see *acl*(5)).

EXAMPLES

In the following example, assume that the file system **/mnt** is to be attached to the file tree at the root directory, (**/**).

This example causes the entire filesystem (**/mnt**) to be dumped on **/dev/rmt/0h** and specifies that the density of the tape is 6250 BPI.

```
/etc/dump 0df 6250 /dev/rmt/0h /mnt
```

DIAGNOSTICS

Many, and verbose.

AUTHOR

dump and **rdump** were developed by the University of California, Berkeley.

FILES

/dev/rdsk/c0d0s0	default filesystem to dump from
/dev/rmt/0m	default tape unit to dump to
/etc/dumpdates	new format-dump-date record
/etc/checklist	dump table: file systems and frequency
/etc/group	used to find group **operator**

SEE ALSO

restore(1M), rmt(1M), acl(5), checklist(4).

NAME

dumpfs - dump file system information

SYNOPSIS

dumpfs *rootdir*
dumpfs *devicefile*

DESCRIPTION

dumpfs prints on the standard output the super block and cylinder group information for the file system with the named root directory or that resides on the named device special file. The listing is very long and detailed. This command is typically used to find certain file system information such as the file system block size and minimum free space percentage.

AUTHOR

dumpfs was developed by the University of California, Berkeley.

SEE ALSO

fsck(1M), mkfs(1M), newfs(1M), tunefs(1M), disktab(5), fs(5).

NAME

ecclogger, eccscrub - check for or scrub out ECC memory errors

SYNOPSIS

ecclogger [*logsize*]
eccscrub

DESCRIPTION

ecclogger checks ECC (error-checking and correcting) memory installed in an HP 9000 Model 350 computer system for errors which have been corrected. When **ecclogger** finds an error, it writes a record of the error to **/etc/ecclog** and invokes **eccscrub** to ensure that all errors have been cleared (only one error per ECC memory card is logged).

The optional parameter *logsize* specifies the number of log entries which are allowed. Hewlett-Packard recommends that the default value of 100 be used for *logsize*. If 100 errors occur on a system the local HP Sales and Support Office should be notified so that the **/etc/ecclog** can be evaluated (a system with fewer than 100 ecclog entries is usually not a concern). When the log reaches the maximum number of entries, a message is printed (mailed to root when invoked from root's **crontab**) to indicate that **/etc/ecclog** is full.

ecclogger should be invoked by root's **crontab** by including an entry for **ecclogger** in the **crontab** file for user **root**. The recommended frequency for running **ecclogger** is once per hour (*eccscrub* is automatically invoked by **ecclogger** whenever an error is corrected).

eccscrub performs a read-modify-write operation on memory to correct any single-bit soft (not a 'stuck' bit) errors that may exist in a memory cell. It may be desirable to invoke **eccscrub** from **cron** once per day.

In order to achieve the recommended frequencies, the following two entries need to be added to the **crontab** entry for root:

```
0 * * * *     exec /etc/ecclogger
0 0 * * *     exec /etc/eccscrub
```

Here is a typical entry for a failure as recorded in **/etc/ecclog**:

```
870911084132 0xFF55CF40 0x70
```

Note: Do not write to **/etc/ecclog**. Doing so prevents **ecclogger** from writing to **/etc/ecclog**.

The first field in the error entry is a date/time stamp [yymmddhhmmss] that indicates when the error was logged. The second field is the memory location in hexadecimal. The final field is the error syndrome byte. The syndrome byte is decoded below (useful to service personell when troubleshooting ECC cards):

Syndrome	Error Location	Syndrome	Error Location
0x01	check bit 0	0x31	data bit 14
0x02	check bit 1	0x34	data bit 15
0x04	check bit 2	0x40	check bit 6
0x08	check bit 3	0x4A	data bit 1
0x0B	data bit 17	0x4F	data bit 0
0x0E	data bit 16	0x52	data bit 2
0x10	check bit 4	0x54	data bit 3
0x13	data bit 18	0x57	data bit 4
0x15	data bit 19	0x58	data bit 5
0x16	data bit 20	0x5B	data bit 6
0x19	data bit 21	0x5D	data bit 7
0x1A	data bit 22	0x62	data bit 24
0x1C	data bit 23	0x64	data bit 25
0x20	check bit 5	0x67	data bit 26
0x23	data bit 8	0x68	data bit 27
0x25	data bit 9	0x6B	data bit 28
0x26	data bit 10	0x6D	data bit 29
0x29	data bit 11	0x70	data bit 30
0x2A	data bit 12	0x75	data bit 31
0x2C	data bit 13		

FILES

`/etc/ecclog`

SEE ALSO

cron(1M), crontab(1)

NAME

edquota - edit user disk quotas

SYNOPSIS

/etc/edquota [**-p** *proto-user*] *username* ...
/etc/edquota -t

Remarks

When establishing quotas for a user who has had none before, the quota statistics for that user will not include any currently occupied file system resources. See WARNINGS below.

DESCRIPTION

edquota is the quota editor. One or more users can be specified on the command line. For each *username*, a temporary file is created with a textual representation of the current disk quotas for that user, and an editor is then invoked on the file. The quotas can then be modified, new quotas added, etc. Upon leaving the editor, *edquota* reads the temporary file and modifies the binary quota files to reflect the changes made.

The editor invoked is *vi*(1) unless the EDITOR environment variable specifies otherwise.

Only users who have appropriate privileges can edit quotas. (In order for quotas to be established on a file system, the root directory of the file system must contain a file called **quotas**. See *quota*(5) for details.)

Options

-p *proto_user* Duplicate the quotas of *proto_user* for each *username*. This is the normal mechanism used to initialize quotas for groups of users.

-t Edit the time limits for each file system. Time limits are set for file systems, not users. When a user exceeds the *soft* limit for blocks or inodes on a file system, a count-down timer is started and the user has an amount of time equal to the time limit in which to reduce usage to below the soft limit (required action is given by the *quota* command). If the time limit expires before corrective action is taken, the quota system enforces policy as if the *hard* limit had been exceeded. The default time limit of 0 is interpreted to mean the value in **<sys/quota.h>**, or one week (7 days). Time units of sec(onds), min(utes), hour(s), day(s), week(s), and month(s) are understood. Time limits are printed in the greatest possible time unit such that the value is greater than or equal to one.

Temporary File Formats

Here is an example of the temporary file created for editing user block and inode quotas:

```
fs /mnt blocks (soft = 100, hard = 120) inodes (soft = 0, hard = 0)
fs / blocks (soft = 1000, hard = 1200) inodes (soft = 200, hard = 200)
```

Here is the format for editing quota time limits:

```
fs /mnt blocks time limit = 10.00 days, files time limit = 20.00 days
fs / blocks time limit = 0 (default), files time limit = 0 (default)
```

When editing *(default)* values, it is not necessary to remove the *(default)* string. For example, to change the **blocks time limit** for /, changing the **0** to **4 days** is sufficient.

WARNINGS

When establishing non-zero quotas for a user who has had none before (for either blocks or inodes), it is necessary to run *quotacheck*(1M) to collect statistics for that user's current usage of that file system. See *quota*(5) for a detailed discussion of this topic.

AUTHOR

edquota was developed by the University of California, Berkeley, and by Sun Microsystems, Inc.

FILES

/etc/checklist	default file systems
/etc/mnttab	information about mounted file systems
directory/quotas	quota statistics static storage for file system where *directory* is the root of the file system specified to *mount*(1M).

SEE ALSO
vi(1), quota(5) privilege(5).

NAME

eisa_config - EISA configuration tool

SYNOPSIS

eisa_config

eisa_config [**-a**]

eisa_config [**-c** *cfgfile*]

eisa_config [**-n** *scifile*]

DESCRIPTION

eisa_config is a specialized program for configuring EISA and ISA (referred to collectively as E/ISA) I/O boards on HP-UX workstations equipped with EISA backplanes. It is used each time the E/ISA configuration is to be changed in any way; i.e., whenever an EISA or ISA board is added to the system, removed from the system, or moved to a different location in the system. **eisa_config** should be run before any physical board configuration or installation changes are made. (This is not necessary in some cases -- see automatic mode below.)

eisa_config interprets information stored in configuration files and uses it to configure system resources needed to properly interact with E/ISA boards. Even though they may be physically present in the computer, E/ISA boards cannot be used by the HP-UX operating system until configuration by **eisa_config** is complete.

The **eisa_config** command takes one of four forms:

eisa_config	Use interactive commands to examine or modify configuration. **eisa_config** prompts for a command, executes it, reports the results of command execution, then prompts for the next command.
eisa_config -a	Attempt to automatically add new EISA boards to the configuration. This option is used by **/etc/bcheckrc** but should not be used elsewhere. ISA boards cannot be added with this option.
eisa_config -c *cfgfile*	Check configuration (CFG) file (discussed below). This option is used mostly by E/ISA board developers. It simply checks the specified CFG file to verify that it follows correct grammar and can be used by **eisa_config**. This option does not affect current configuration in any way.
eisa_config -n *scifile*	Non-target mode. This option uses the contents of *scifile* instead of non-volatile memory (NVM) to set up E/ISA configuration, and is most commonly used for creating identical configurations on multiple workstations.

Assigning Resources

Depending on their design, internal capabilities, and their role in system operation, E/ISA boards use various combinations of one or more system resources such as DMA channels, interrupt lines, memory, etc. Also, given boards do not always use a full set of system resources; for example, EISA provides 11 interrupt lines, but a given board might be able to use only lines 3, 5, and 6. Thus a means for the board to determine what resources are to be used must be provided.

ISA boards use physical switches or jumpers on the board to specify what resources are to be used. The person installing the board sets the switches or jumpers as specified by the board's manufacturer and based on system needs. There are thousands of different kinds of ISA boards, but unfortunately there are no standard conventions for switch and jumper usage. This results in much confusion and numerous configuration problems. For example, it is easy to inadvertently assign a given resource to two different boards, but often very difficult to diagnose the problem.

EISA boards usually have no switches or jumpers for resource assignment. Instead, each EISA board has a corresponding configuration (CFG) file that tells the system how the board can be used and what resources it needs. **eisa_config** is the HP-UX system program that interprets the various CFG files for all boards in the system, then builds a conflict-free configuration.

Configuration Files

All EISA boards have a corresponding CFG file. ISA boards, when used in HP-UX systems, must also have a

corresponding CFG file. Although **eisa_config** cannot automatically configure an ISA board, it can use the contents of the CFG file to determine what switch or jumper settings on an ISA board can be used to prevent resource conflicts.

eisa_config expects to find a CFG file for each E/ISA board connected to the workstation. The administrator is responsible for making sure that these CFG files are present in directory **/etc/eisa**. CFG files corresponding to boards being used should always be kept in this directory. Do not remove them after **eisa_config** is run the first time, because they will be needed every time the configuration is changed, such as when a new board is added or one is removed. Do not change the file names of the CFG files. The file name has a specific format which is used by **eisa_config** to automatically match a board with its CFG file.

CFG files are normally supplied by the E/ISA board manufacturer. Two scenarios apply:

- If the E/ISA board is supplied by HP, the CFG file corresponding to the board is loaded into **/etc/eisa** as part of normal operating system installation. It should never be removed.
- If the E/ISA board is not supplied by HP, install both the CFG file and the software driver for the board from HP-UX-readable media supplied by the board manufacturer. Copy the CFG file to directory **/etc/eisa** where it must remain as long as the card is present in the system.

All CFG files must follow a grammar specified in the EISA bus specification. The most basic building block in the CFG grammar is the *board*. Each board has several attributes including board ID (to match with a board's ID register), manufacturer, ASCII text describing what the board does, what kinds of slots the board can go in, whether the board has a readable ID register, and various other capability attributes.

Each file can also contain lists of board-wide resources (such as I/O registers, switches, and jumpers) and how they should be initialized.

A board can be treated as a set of one or more *functions* where a given board contains a single function or multiple functions. An example of a two-function board is one having both a serial port and a parallel printer port. Each function has a separate block in that board's CFG file. Each function has a name, a type, and a set of configuration *choices*.

Each *choice* block has a name and a set of attributes. These attributes include what resources the choice requires and whether the function is enabled or disabled by that choice. Initialization is also usually specified within a choice. A given choice might require that certain registers be initialized to a specified value and that switches be set in a certain way.

Configuration Processing

E/ISA configuration is handled as follows:

- **eisa_config** builds a conflict-free configuration, then saves the configuration in EISA non-volatile memory (NVM).
- Appropriate drivers and device files must be installed before rebooting the system.
- Next time the operating system is rebooted, the HP-UX kernel initializes the specified E/ISA boards according to the contents of NVM.

If a board is currently present in the system, but has no corresponding configuration data in NVM, the EISA board cannot be used until the **eisa_config** program is run again and the new board is accounted for in NVM. A newly installed or existing E/ISA board is not usable until **eisa_config** has added it and the system has been rebooted with the necessary drivers and device special files installed. See EXAMPLES for an illustration of how to add a new board to the system.

It is possible to add EISA boards that do not have switches or jumpers to the configuration without running **eisa_config** interactively. The **/etc/bcheckrc** script invokes **eisa_config** with automatic mode during each system initialization. If a board has been added since the last time **eisa_config** was executed, **eisa_config** attempts to add the new board to the configuration. If the new board is successfully added, the system may need to be rebooted (**/etc/bcheckrc** does this automatically). If the new board could not be added to the configuration, a warning is written to the system console and **/etc/eisa/config.err**.

In addition to writing to NVM, **eisa_config** also automatically saves the current configuration to an SCI file called **/etc/eisa/system.sci**. SCI files can also be created by the interactive **save** command (see below). The E/ISA subsystem can also be initialized from an SCI file, rather than from NVM by using the

eisa_config -n command form discussed earlier. SCI files are quite useful when a site has several identically-configured workstations. Run **eisa_config** on one system and save the configuration in an SCI file. Copy this file to other systems, then use it to initialize those systems. Remember that the configuration must be saved to NVM and the system rebooted before the E/ISA boards can be used.

Drivers and Device Files

Running **eisa_config** is not the only task necessary when adding an E/ISA board to a system. Corresponding I/O drivers must be added to the kernel and appropriate device files must be created. These steps are the same as is required for any I/O card, and can be performed either before or after running **eisa_config**. The important thing to remember is that the E/ISA board cannot be used until *all* necessary tasks are complete.

Interactive Commands

If the command form **eisa_config** is used, **eisa_config** runs in interactive mode. Interactive mode conducts configuration changes by using a series of keyboard commands. **eisa_config** prompts for a command, executes it, displays the results of executing the command, then prompts for the next command. Interactive commands are broadly grouped into five categories:

action	Alter the configuration in some way.
display	Show current configuration.
cfg	Manage CFG files.
comments	Display help and comments information found in CFG files.
help	Help for using **eisa_config** interactive commands

The *action* commands are:

add *cfgfile slotnum* Adds a board to the current configuration. *cfgfile* specifies which CFG file corresponds to the board and *slotnum* identifies the slot where the board resides.

remove *slotnum* Remove a board from the current configuration. *slotnum* identifies the slot where the board currently resides.

move *curslotnum newslotnum*
Move a board that is currently configured in one slot to a different slot. *curslotnum* and *newslotnum* specify the current and new slot numbers, respectively.

change *slotnum functionnum choicenum*
Change the choice used for a given function. All three arguments, *slotnum*, *functionnum*, and *choicenum* are required. The function number (*functionnum*) and choice number (*choicenum*) can be obtained by using the **show board** command on the slot in question. Function numbers are of the format **F***num* and choice numbers are of the format **CH***num*. Note that a board must already be part of the configuration before the change command can be used.

When **eisa_config** adds a board, it selects a choice for each function. Generally, the first choice for each function is selected (the default). However, in order to resolve conflicts, **eisa_config** may select a different choice for a given function. When specifying a choice for a particular function by use of the **change** command, **eisa_config** always uses that choice; it does not select a different one, even when a conflict needs to be resolved.

save [*filename*] Save the current configuration. If the current configuration is not conflict-free, a warning is produced and the save is not done. If you specify a file name, the save is done to that file; otherwise, the save is done to NVM (and the **/etc/eisa/system.sci** file). Note that the **quit** command also (optionally) saves the configuration to NVM (and file **/etc/eisa/system.sci**).

When the configuration is saved to NVM, a log file is created that provides a brief desription of the new configuration. The log file is named **/etc/eisa/config.log**, and contains information generated by a **show** command, followed by a **show board** command, followed by a **show switch** command.

init [*filename*] Initialize the configuration. The initial configuration is retrieved from a file if one has been specified. Otherwise, it is retrieved from NVM. Note that an implicit **init** is done when **eisa_config** is first started. This command should only be used when the current configuration **eisa_config** is dealing with is incorrect. For example, if you make some changes that you decide you do not want, you can use this command to start over.

quit Leave **eisa_config**. If the configuration is conflict-free and has been changed, you are asked if you want to save the configuration (to NVM). If any switches or jumpers have to be changed as a result of this new configuration, you are notified of these changes prior to saving the configuration. Be sure that all switches and jumpers match what **eisa_config** has specified before booting the system.

When the configuration is saved to NVM, a log file is created that provides a brief desription of the new configuration. The log file is named **/etc/eisa/config.log**, and contains information generated by a **show** command, followed by a **show board** command, followed by a **show switch** command.

The *show* (display) commands are:

show List all slots and their current status; i.e., whether occupied by a particular board, or empty.

show slots *cfgfile*
List all of the slots that could accept the board corresponding to the CFG file *cfgfile*.

show board [*cfgfile* | *slotnum*]
List the basic attributes for the selected board or boards. Includes a list of all the functions on the board and a list of all available choices for each function. If the board is currently part of the configuration, the currently selected choice is marked. The default choice is the first choice listed for each function. If a board is not specified (either by CFG file name or slot number), information is displayed for each of board installed and configured in the system.

show switch [**changed**] [*slotnum*]
List the switch and jumper settings (both default and required) for the boards in the configuration. If the keyword **changed** is used, only those switches and jumpers that were changed from the previous configuration are displayed. If a slot number is specified, only switches and jumpers on the board in that slot are displayed. Note that **show switch** supports all combinations of **changed** and *slotnum*.

There are two kinds of *cfg* commands:

cfgtypes List the types of boards that have CFG files in directory **/etc/eisa** and how many CFG files in **/etc/eisa** are of each type.

cfgfiles [*type*] List all CFG files that are currently available for use in the **/etc/eisa** directory. If a specific board *type* is specified, only CFG files of that type are displayed.

comment commands extract the help and comments text provided in the specified CFG file or files. Both help and comments are displayed if they are available. Each command form accepts as an argument either a CFG file or a slot number identifying which board you want help for.

comment board [*cfgfile* | *slotnum*]
Display board-level help and comments.

comment function [*cfgfile* | *slotnum*]
Display function-level help and comments.

comment choice [*cfgfile* | *slotnum*]
Display choice-level help.

`comment switch` [*cfgfile* | *slotnum*]
Display help and comments for switches and/or jumpers as appropriate.

Note that all arguments (except the type of comments requested) are optional. If no optional argument is specified, all available comments for the specified file or board are extracted. For example:

`comment board 1`
Display help and comments available for the board currently configured in slot 1.

`comment board` Display help and comments available for *all* currently configured boards.

The *help* commands explain how to use the `eisa_config` interactive commands. If no other arguments are given, help is displayed for all of the interactive commands. Alternatively, any valid command can be used as a argument to the help command. Help is then given for the specified command only.

`help` Display a brief explanation of all valid `eisa_config` interactive commands.

`help` [*cmdname*] Display an explanation of the command specified.

EXAMPLES

Add a new E/ISA board to the system:

1. Load the CFG file (from media provided by the manufacturer) into directory `/etc/eisa` if the file is not already present.
2. Run `eisa_config`. `eisa_config` reads the contents of NVM to obtain current system configuration.
3. Use the interactive `add` command to add the new board. `eisa_config` reads the corresponding CFG file to obtain needed configuration information.
4. Exit `eisa_config`, noting any required switch or jumper settings. `eisa_config` generates a new configuration and writes it to NVM. The required switch and jumper settings are also saved in the log file `/etc/eisa/config.log`.
5. Add the correct software drivers for the board (and board devices) to the kernel, and use *mknod*(1M) to create any needed device special files.
6. Shut down and disconnect power to the system.
7. Install the E/ISA board after changing any switch or jumper settings required by `eisa_config`.
8. Reboot the system. When the system is running again, the contents of NVM will match the E/ISA boards present in the system, and the newly added board can be used immediately.

This procedure can also be used to add multiple new boards at the same time. Simply use the `add` command once for each board and alter the other steps as appropriate.

If the board to be added is an EISA board that does not have switches or jumpers, the board can be added via automatic mode; that is, steps 2-4 above can be skipped.

AUTHOR

`eisa_config` was developed by HP and Compaq.

FILES

`/etc/eisa/!XXX0000.CFG`	CFG files
`/etc/eisa/config.err`	errors encountered in automatic mode
`/etc/eisa/config.log`	log file containing current E/ISA configuration
`/etc/eisa/system.sci`	mirror image of configuration saved to NVM

SEE ALSO

config(1M), mknod(1M).

NAME

envd - system physical environment daemon

SYNOPSIS

/etc/envd [**-f** *configfile*]

DESCRIPTION

The **envd** daemon provides a means for the system to respond to environmental conditions detected by hardware. Such responses are typically designed to maintain file system integrity and prevent data loss. The environmental condition currently recognized by **envd** is over-temperature.

envd logs messages and then executes actions when a supported environmental event is detected. Whether to do message logging and what actions to perform for a given environmental event are determined by *configfile* (default is **/etc/envd.conf**). If no **-f** option was specified and the default configfile **/etc/envd.conf** does not exist, **envd** fails. A recommended default *configfile* is available in **/etc/newconfig/envd.conf**. The *configfile* (or **/etc/envd.conf**) is only examined when the daemon is started or when it receives a **SIGHUP** signal to restart and re-initialize the daemon itself.

envd uses the **syslog** message logging facility to log warning messages. If *configfile* specifies messages to be logged, the destination of the warning messages is determined by the configuration of the **LOG_DAEMON** facility of the **syslogd** daemon (see *syslogd*(1M) and *syslog*(3C) for details) and various **syslog** priorities defined below for the corresponding environmental events. Warning messages are written to the console if **envd** is unable to send to **syslogd**.

The *configfile* is composed of event lines, each of which followed by zero or more action lines. Comment lines can be interspersed at any point. No more than one event line can be specified for a given event.

Event — Event lines consist of an event keyword and a message indicator, separated by a colon (**:**). Valid event keywords are **OVERTEMP_CRIT** and **OVERTEMP_EMERG**. Valid message indicators are **y** and **n**. An example is **OVERTEMP_EMERG:y**, indicating that warning messages are to be sent for the OVERTEMP_EMERG event.

Event keywords must start in the first column, and only one event and one message indicator are allowed on a given line.

Action — Action lines can consist of a sequence of any valid **/bin/sh** commands or pipelines. Lines from one event line to the next event line, or to the end of the file, are part of the action lines for the preceding event, and are passed intact to the shell to execute upon detecting the event. The action for an event can span across several lines, but the syntax of every line must be understood by **/bin/sh**. There are no default actions for any events if no action lines are specified.

No parsing or syntax checking is performed on the action lines; system administrators are responsible for verifying the correctness of the action syntax.

Comments — Lines beginning with the **#** character in the first column are comment lines, and all characters up to the subsequent new-line character are ignored.

Blank lines are ignored as comment lines.

Here is an example **/etc/envd.conf** file:

```
# The example below configures envd to log the warning message and
# to rcp critical applications to a remote machine at OVERTEMP_CRIT.
# It configures envd to log emergency messages and to perform
# system shutdown at OVERTEMP_EMERG, in order to preserve
# the data integrity.

OVERTEMP_CRIT:y
    /usr/bin/rcp critical_appl_files \
        remote_machine:/backup

OVERTEMP_EMERG:y
    /etc/reboot -qh
```

Only users with appropriate privileges can invoke **envd**.

Over-temperature Handling

Over-temperature handling is supported only on systems equipped with over-temperature sensing hardware. Over-temperature limits may vary, depending on the hardware. Each system processor defines its own safest threshold for supported equipment combinations. The table below shows four levels of temperature states. For the temperature range specific to your system configuration, refer to any of the following documents for your system: *Site Planning and Preparation Guide*, *Installation and Configuration Guide*, or *Operator Handbook*.

State	State Description
NORMAL	Within normal operating temperature range
OVERTEMP_CRIT	Temperature has exceed the normal operating range of the system, but it is still within the operating limit of the hardware media.
OVERTEMP_EMERG	Temperature has exceeded the maximum specified operating limit of hardware media; power loss is imminent. A minimum of about 60 seconds is guaranteed between the OVERTEMP_MID state and the OVERTEMP_POWERLOSS (power loss) state.
OVERTEMP_POWERLOSS	Hardware will disconnect all power from all cards in the system chassis.

The `syslog` priorities mapped to two over-temperature events are: `LOG_EMERG` (for `OVERTEMP_EMERG`) and `LOG_CRIT` (for `OVERTEMP_CRIT`).

Any non-shutdown activities (e.g. file transfer) should be performed at `OVERTEMP_CRIT`. It is important to configure only critical activities for `OVERTEMP_CRIT` because the over-temperature might rise dramatically fast to `OVERTEMP_EMERG`. It is recommended to perform a quick shutdown using `/etc/reboot -qh` at `OVERTEMP_EMERG` to preserve file system data integrity. If the hardware enters the `OVERTEMP_POWERLOSS` state and the system has not been shut down, the sudden loss of power could result in data loss. Note that power-fail recovery functionality is not available in this case. When the hardware powers down, no warning messages are produced, and no action is taken by the system.

Whenever the temperature rises from one level to another (such as from `NORMAL` to `OVERTEMP_CRIT` or from `OVERTEMP_CRIT` to `OVERTEMP_EMERG`, the warning message, if specified, and the corresponding specified over-temperature action is executed once, and only once, per state change.

AUTHOR

`envd` was developed by HP.

FILES

`/etc/envd`	`envd` executable file
`/etc/envd.conf`	default `envd` configuration file
`/etc/syslog.conf`	default `syslog` configuration file
`/etc/envd.action[123]`	`envd` work files

SEE ALSO

reboot(1M), shutdown(1M), syslogd(1M), syslog(3C).

HP-UX System Administration manuals.

NAME

exportfs - export and unexport directories to NFS clients

SYNOPSIS

```
/usr/etc/exportfs
/usr/etc/exportfs -a[v]
/usr/etc/exportfs [-auv]
/usr/etc/exportfs [-uv][dir [dir ...]]
/usr/etc/exportfs -i[v][-o options][dir [dir ...]]
```

DESCRIPTION

exportfs makes a local directory or file available to NFS clients for mounting over the network (directories and files cannot be NFS mounted unless they are first exported by **exportfs**).

exportfs is normally invoked at boot time by the **/etc/netnfsrc** script, and uses information contained in the **/etc/exports** file to export *dir* (which must be specified as a full pathname). The superuser can run **exportfs** at any time to alter the list or characteristics of exported directories and files.

If no options or arguments are specified in the command line, **exportfs** prints a list of currently exported directories and files to the standard output.

Options

exportfs recognizes the following options:

-a All. Export all directories listed in **/etc/exports**, or if **-u** is also specified, unexport all of the currently exported directories.

-i Ignore the options in **/etc/exports**. Normally, **exportfs** consults **/etc/exports** for the options associated with the exported directory.

-u Unexport the indicated directories.

-v Verbose. Print each directory or filename as it is exported or unexported.

-o *options* Specify a comma-separated list of optional characteristics for the directory being exported. *options* can be selected from among:

async All mounts will be asynchronous. Refer to *exports*(4) for warnings when using this option.

ro Export the directory read-only. If not specified, the directory is exported read-write.

rw=*hostname*[**:***hostname*]...
Export the directory read-mostly. Read-mostly means exported read-only to most machines, but read-write to those specified. If not specified, the directory is exported read-write to all. Up to 256 *hostname*s can be specified.

anon=*uid* If a request comes from an unknown user, use *uid* as the effective user ID.

Note: root users (uid*0)* are always treated as user **unknown** by the NFS server unless they are included in the **root** option below.

If the client is a UNIX system, only root users are considered **unknown**. All other users are recognized even if they are not in **/etc/passwd.**

The default value for *uid* is the UID of user **nobody**. If user **nobody** does not exist, the value 65 534 is used. Setting the value of **anon** to 65 535 disables anonymous access.

root=*hostname*[**:***hostname*]...
Give root access only to the root users from a specified *hostname*. The default is for no hosts to be granted root access. Up to 256 *hostname*s can be specified.

access=*client*[:*client*]...
Give mount access to each *client* listed. A *client* can either be a hostname, or a netgroup (see *netgroup*(4)). **exportfs** checks for each *client* in the list first in file **/etc/hosts**, then in **/etc/netgroup**. The default value allows any machine to mount the given directory.

DIAGNOSTICS

If an NFS mounted directory is unexported by **exportfs**, any access by the client to the directory causes an **NFS stale file handle** error. However, if **exportfs** is used to remove a client from the access list of an exported directory, an **NFS stale file handle** error does not result from any access by the client to the directory.

EXAMPLES

List currently exported directories and files:

```
exportfs
```

Export entries in **/etc/exports**

```
exportfs -a
```

Unexport all exported files and directories:

```
exportfs -ua
```

Unexport all exported files and directories and print each directory or filename as it is unexported:

```
exportfs -uav
```

Export **/usr** to the world, ignoring options in **/etc/exports**:

```
exportfs -i /usr
```

Export **/usr/bin** and **/usr/adm** read-only to the world:

```
exportfs -i -o ro /usr/bin /usr/adm
```

Export **/usr/bin** read-write only to systems **polk** and **vanness**:

```
exportfs -i -o rw=polk:vanness /usr/bin
```

Export root access on **/usr/adm** only to the system named **pine**, and mount access to both **pine** and **geary**:

```
exportfs -i -o root=pine, access=pine:geary /usr/adm
```

FILES

/etc/exports	static export information
/etc/xtab	current state of exported directories
/etc/netgroup	list of network groups

SEE ALSO

showmount(1M), exports(4), netgroup(4).

WARNINGS

You cannot export a directory that resides *within the same file system* and is either a parent or sub-directory of a directory that is currently exported. For example, **/usr** and **/usr/local** cannot both be exported if they reside in the same disk partition.

If you unexport a directory, remove a client from the access list, then export again, the client still has access to the directory until the client unmounts the directory. Removing a client from the **root** or **rw** list takes effect immediately.

/etc/xtab is a system file that lists currently exported directories and files. This file is maintained by **exportfs**. To ensure that this file is always synchronous with current system data structures, do not attempt to edit it by hand.

NAME

extendfs - extend file system size

SYNOPSIS

/etc/extendfs [**-q**] [**-v**] [**-s** *size*] *special*

DESCRIPTION

If the original **hfs** filesystem image created on *special* does not make use of all of the available space, **extendfs** can be used to increase the capacity of an *hfs* filesystem by updating the filesystem structure to include the extra space.

The command-line parameter *special* specifies the character device special file of either a logical volume or a disk partition. If *special* refers to a mounted filesystem, *special* must be un-mounted before **extendfs** can be run (see *mount*(1M)).

Options

extendfs recognizes the following options:

-q Query the size of *special*. No file system extension will be done.

-v Verbose flag.

-s*size* Specifies the number of **DEV_BSIZE** blocks to be added to the file system. If **-s***size* is not specified, the maximum possible size is used.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **lvchange** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

To increase the capacity of a filesystem created on a logical volume, enter:

```
umount /dev/vg00/lvol1
lvextend -L larger_size /dev/vg00/lvol1
extendfs /dev/vg00/rlvol1
mount /dev/vg00/lvol1 mount_directory
```

WARNINGS

The root filesystem cannot be extended using the **extendfs** command because the root filesystem is always mounted, and **extendfs** only works on unmounted filesystems.

RETURN VALUE

extendfs returns the following values:

0 No errors were detected and filesystem was successfully extended.

1 Command aborted.

REMARKS

In order to extend a filesystem to the 4-Gbyte maximum filesystem size, enough Cylinder Group summary blocks must be reserved by **mkfs** (see *mkfs*(1M)). For a filesystem created by an older version of **mkfs** (prior to HP-UX Release 9.0 — see *mkfs*(1M)), the full 4-Gbyte filesystem capacity may not be achievable by use of **extendfs**.

SEE ALSO

lvextend(1M), mkfs(1M), mount(1M), fs(4).

NAME

fbackup - selectively back up files

SYNOPSIS

/etc/fbackup -f *device* [**-f** *device*] ... [**-0-9**] [**-uvyAH**] [**-i** *path*] [**-e** *path*] [**-g** *graph*] [**-I** *path*] [**-V** *path*] [**-c** *config*]

/etc/fbackup -f *device* [**-f** *device*] ... [**-R** *restart*] [**-uvyAH**] [**-I** *path*] [**-V** *path*] [**-c** *config*]

DESCRIPTION

fbackup combines features of **dump** and **ftio** to provide a flexible, high-speed file system backup mechanism (see *dump*(1M) and *ftio*(1)). **fbackup** selectively transfers files to an output device. For each file transferred, the file's contents and all the relevant information necessary to restore it to an equivalent state are copied to the output device. The output device can be a raw magnetic tape drive, the standard output, a DDS-format tape, a rewritable magneto-optical disk or a file.

The selection of files to backup is done by explicitly specifying trees of files to be included or excluded from an **fbackup** session. The user can construct an arbitrary graph of files by using the **-i** or **-e** options on the command line, or by using the **-g** option with a graph file. For backups being done on a regular basis, the **-g** option provides an easier interface for controlling the backup graph. **fbackup** selects files in this graph, and attempts to transfer them to the output device. The selectivity depends on the mode in which **fbackup** is being used; i.e., full or incremental backup.

When doing full backups, all files in the graph are selected. When doing incremental backups, only files in the graph that have been modified since a previous backup of that graph are selected. If an incremental backup is being done at level 4 and the **-g** option is used, the database file is searched for the most recent previous backup at levels 0-3. If a file's modification time is before the time when the last appropriate session began and the i-node change time is before the time that same session ended, the file is not backed up. Beginning at HP-UX Release 8.0, all directories lying on the path to a file that qualifies for the incremental backup will also be on the backup media, even if the directories do not qualify on their own status.

If **fbackup** is used for incremental backups, a database of past backups must be kept. **fbackup** maintains this data in the text file **/usr/adm/fbackupfiles/dates**, by default. Note that the directory **/usr/adm/fbackupfiles** must be created prior to the first time **fbackup** is used for incremental backups. The **-d** option can be used to specify an alternate database file. The user can specify to update this file when an **fbackup** session completes successfully. Entries for each session are recorded on separate pairs of lines. The following four items appear on the first line of each pair: the graph file name, backup level, starting time, and ending time (both in *time*(2) format). The second line of each pair contains the same two times, but in *strftime*(3C) format. These lines contain the local equivalent of **STARTED:**, the start time, the local equivalent of **ENDED:**, and the ending time. These second lines serve only to make the dates file more readable; **fbackup** does not use them. All fields are separated by white space. Graph file names are compared character-by-character when checking the previous-backup database file to ascertain when a previous session was run for that graph. Caution must be exercised to ensure that, for example, **graph** and **./graph** are not used to specify the same graph file because **fbackup** treats them as two different graph files.

The general structure of a **fbackup** volume is the same, no matter what type of device is used. There are some small specific differences due to differing capabilities of devices. The general structure is as follows:

- Reserved space for ASCII tape label (1024 bytes)
- **fbackup** specific volume label (2048 bytes)
- session index (size in field of volume label)
- data

Each file entry in the index contains the volume number and the pathname of the file. At the beginning of every volume, **fbackup** assumes that all files not already backed up will fit on that volume; an erroneous assumption for all but the last volume. Indices are accurate only for the previous volumes in the same set. Hence, the index on the last volume may indicate that a file resides on that volume, but it may not have actually been backed up (for example, if it was removed after the index was created, but before **fbackup** attempted to back it up). The only index guaranteed to be correct in all cases is the on-line index (**-I** option), which is produced after the last volume has been written. Specific minor differences are listed below:

- When using 9-track tape drives or DDS-format tape drives several small differences exist. The main blocks of information are separated by EOFs. **fbackup** checkpoints the media periodically to enhance error recovery. If a write error is detected, the user normally has two options: First, a new volume can be mounted and that volume rewritten from the beginning. Second, if the volume is not too severely damaged, the good data before the error can be saved, and the write error is treated as a normal end-of-media condition. The blocks of data with their checkpoint records are also separated by EOFs. In addition if the DDS-format drive supports *Fast Search Marks* these will be used to enhance recovery speed by placing them between blocks of files.
- For a magneto-optical device, a disk, a file, or standard output, there are no special marks separating the information pieces. Using standard output results in only one volume.

fbackup provides the ability to use UCB-mode tape drives. This makes it possible to overlap the tape rewind times if two or more tape drives are connected to the system.

Set-up

There are several things the user will want to consider when setting **fbackup** up for regular use. These include type of device and media, full versus incremental frequency, amount of logging information to keep on-line, structure of the graph file, and on-line versus off-line backup.

The type of device used for backups can affect such things as media expenses, ability to do unattended backup and speed of the backup. Using 9-track tapes will probably result in the highest performance, but require user intervention for changing tapes. A magneto-optical autochanger can provide an unattended backup for a large system and long life media, however the media cost is high. The lowest cost will probably be achieved through DDS-format devices, but at the lowest performance.

It is also important to consider how often full backups should be made, and how many incremental backups to make between full backups. Time periods can be used, such as a full backup every Friday and incrementals on all other days. Media capacities can be used if incremental backups need to run unattended. The availability of personnel to change media can also be an important factor as well as the length of time needed for the backup. Other factors may affect the need for full and incremental backup combinations such as contractual or legal requirements.

If backup information is kept online; i.e., output from the **-V** or **-I** options, the required storage space must also be considered. Index file sizes are hard to predict in advance because they depend on system configuration. Each volume header file takes less than 1536 bytes. Of course the more information that is kept on-line, the faster locating a backup media for a recovery will be. Another point to consider is that the default directory for the database file is **/usr/adm** and this is a context-dependent file on clustered systems, meaning that files may or may not be visible, depending on system context.

There are several ways to structure the graph file or files used in a system backup. The first decision involves whether to use one or more than one graph files for the backup. Using one file is simpler, but less flexible. Using two or more graph files simplifies splitting backups into logical sets. For example, one graph file can be used for system disks where changes tend to be less frequent, and another graph file for the users area. Thus two different policies can be implemented for full and incremental backups.

fbackup was designed to allow backups while the system is in use by providing the capability to retry an active file. When absolute consistency on a full backup is important, the system should probably be in single-user mode. However, incremental backups can be made while the system is in normal use, thus improving system up-time.

Options

-c *config* *config* is the name of the configuration file, and can contain values for the following parameters:

- Number of 1024-byte blocks per record,
- Number of records of shared memory to allocate,
- Number of records between checkpoints,
- Number of file-reader processes,
- Maximum number of times **fbackup** is to retry an active file,
- Maximum number of bytes of media to use while retrying the backup of an active file,
- Maximum number of times a magnetic tape volume can be used,

- Name of a file to be executed when a volume change occurs. This file must exist and be executable.
- Name of a file to be executed when a fatal error occurs. This file must exist and be executable.
- The number of files between the *Fast Search Marks* on DDS-format tapes. The cost of these marks are negligible in terms of space on the DDS-format tape. Not all DDS-format devices support fast search marks.

Each entry in the configuration file consists of one line of text in the following format: identifier, white space, argument. In the following sample configuration file, the number of blocks per record is set to 16, the number of records is set to 32, the checkpoint frequency is set to 32, the number of file reader processes is set to 2, the maximum number of retries is set to 5, the maximum retry space for active files is set to 5,000,000 bytes, the maximum number of times a magnetic tape volume can be used is set to 100, the file to be executed at volume change time is **/usr/adm/fbackupfiles/chgvol**, the file to be executed when a fatal error occurs is **/usr/adm/fbackupfiles/error**, and the number of files between fast search marks is set to 200.

```
blocksperrecord     16
records             32
checkpointfreq      32
readerprocesses     2 (maximum of 6)
maxretries          5
retrylimit          5000000
maxvoluses          100
chgvol              /usr/adm/fbackupfiles/chgvol
error               /usr/adm/fbackupfiles/error
filesperfsm         200
```

Each value listed is also the default value, except **chgvol** and **error**, which default to null values.

-d *path* This specifies a path to a database for use with incremental backups. It overrides the default database file **/usr/adm/fbackupfiles/dates**.

-e *path* *path* specifies a tree to be excluded from the backup graph. This tree must be a subtree of part of the backup graph. Otherwise, specifying it will not exclude any files from the graph. There is no limit on how many times the **-e** option can be specified.

-f *device* *device* specifies the name of an output file. If the name of the file is **-**, **fbackup** writes to the standard output. There is no default output file; at least one must be specified. If more than one output file is specified, **fbackup** uses each one successively and then repeats in a cyclical pattern. Patterns can be used in the device name in a manner resembling file name expansion as done by the shell (see *sh-bourne*(1) and other shell manual entries. The patterns must be protected from expansion by the shell by quoting them. The expansion of the pattern results in all matching names being in the list of devices used.

There is slightly different behavior if remote devices are used. A device on the remote machine can be specified in the form *machine*:*device*. **fbackup** creates a server, **/etc/rmt**, on the remote machine to access the tape device. The pattern matching capability does not apply to remote devices. Only half-inch 9-track magnetic tapes or DDS-format tapes can be remote devices. The fast search capability is not used when remote DDS-format devices are used.

-g *graph* *graph* defines the graph file. The graph file is a text file containing the list of file names of trees to be included or excluded from the backup graph. These trees are interpreted in the same manner as when they are specified with the **-i** and **-e** options. Graph file entries consist of a line beginning with either **i** or **e**, followed by white space, and then the path name of a tree. Lines not beginning with **i** or **e** are treated as an error. There is no default graph file. For example, to backup all of **/usr** except for the subtree **/usr/lib**, a file could be created with the following two records:

```
i /usr
e /usr/lib
```

`-i` *path* *path* specifies a tree to be included in the backup graph. There is no limit on how many times the `-i` option can be specified.

`-n` Cross NFS mount points. By default `fbackup` does not cross NFS mount points, regardless of paths specified by the `-i` or `-g` options.

`-s` Backup the object that a symbolic link refers to. The default behavior is to backup the symbolic link.

`-u` Update the database of past backups so that it contains the backup level, the time of the beginning and end of the session, and the graph file used for this `fbackup` session. For this update to take place, the following conditions must exist: Neither the `-i` nor the `-e` option can be used; the `-g` option must be specified exactly once (see below); the `fbackup` must complete successfully.

`-v` Run in verbose mode. Generates status messages that are otherwise not seen.

`-y` Automatically answer `yes` to any inquiries.

`-A` Do not back up optional entries of access control lists (ACLs) for files. Normally, all mode information is backed up including the optional ACL entries. With the `-A` option, the summary mode information (as returned by `stat()`) is backed up. Use this option when backing up files from a system that contains ACLs to be recovered on a system that does not understand ACLs (see *acl*(5)).

`-H` Search hidden subdirectories (context-dependent files or CDFs). Normally, only the CDF element matching the current context is backed up without expanding the path name to show the actual element. For more information on CDFs, see *cdf*(4).

`-I` *path* *path* specifies the name of the on-line index file to be generated. It consists of one line for each file backed up during the session. Each line contains the volume number on which that file resides and the file name. If the `-I` option is omitted, no index file is generated.

`-V` *path* The volume header information is written to *path* at the end of a successful `fbackup` session. The following fields from the header are written in the format *label*`:`*value* with one pair per line.

`Magic Field` On a valid `fbackup` media it contains the value `FBACKUP LABEL`.

`Machine Identification` This field contains the result of `uname -m`.

`System Identification` This field contains the result of `uname -s`.

`Release Identification` This field contains the result of `uname -r`.

`Node Identification` This field contains the result of `uname -n`.

`User Identification` This field contains the result of `cuserid()` (see *cuserid*(3S)).

`Record Size` This field contains the maximum length in bytes of a data record.

`Time` This field contains the clock time when `fbackup` was started.

`Media Use` This field contains the number of times the media has been used for backup. Since the information is actually on the media, this field will always contain the value 0.

`Volume Number` This field contains a `#` character followed by 3 digits, and identifies the number of volumes in the backup.

`Checkpoint Frequency` This field contains the frequency of backup-data-record checkpointing.

`Index Size` This field contains the size of the index.

`Backup Identification Tag`
: This field is composed of two items: the process ID (pid) and the start time of that process.

`Language`
: This field contains the language used to make the backup.

`-R` *restart*
: Restart an `fbackup` session from where it was previously interrupted. The *restart* file contains all the information necessary to restart the interrupted session. None of the `-[ieg0-9]` options can be used together with the restart option.

`-0-9`
: This single-digit number is the backup level. Level `0` indicates a full backup. Higher levels are generally used to perform incremental backups. When doing an incremental backup of a particular graph at a particular level, the database of past backups is searched to find the date of the most recent backup of the same graph that was done at a lower level. If no such entry is found, the beginning of time is assumed. All files in the graph that have been modified since this date are backed up.

Access Control Lists (ACLs)

If a file has optional ACL entries, the **-A** option is required to enable its recovery on a system whose access control lists capability is not present.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the order in which files are stored in the backup device and the order output by the `-I` option.

`LC_TIME` determines the format and contents of date and time strings.

`LANG` determines the language in which messages are displayed.

If `LC_COLLATE` and `LC_TIME` are not both specified in the environment or if either is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `fbackup` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

`fbackup` returns 0 upon normal completion, 1 if it is interrupted but allowed to save its state for possible restart, and 2 if any error conditions prevent the session from completing.

EXAMPLES

In the following two examples, assume the graph of interest specifies all of `/usr` except `/usr/lib` (as described in the `g` key section above).

The first example is a simple case where a full backup is done but the database file is not updated. This can be invoked as follows:

```
/etc/fbackup -0i /usr -e /usr/lib -f /dev/rmt/0h
```

The second example is more complicated, and assumes the user wants to maintain a database of past `fbackup` sessions so that incremental backups are possible.

If sufficient on-line storage is available, it may be desirable to keep several of the most recent index files on disk. This eliminates the need to recover the index from the backup media to determine if the files to be recovered are on that set. One method of maintaining on-line index files is outlined below. The system administrator must do the following once before `fbackup` is run for the first time (creating intermediate level directories where necessary):

- Create a suitable configuration file called `config` in the directory `/usr/adm/fbackupfiles`
- Create a graph file called `usr-usrlib` in the directory `/usr/adm/fbackupfiles/graphs`
- Create a directory called `usr-usrlib` in the directory `/usr/adm/fbackupfiles/indices`

A shell script that performs the following tasks could be run for each **fbackup** session:

- Build an index file path name based on both the graph file used (passed as a parameter to the script) and the start time of the session (obtained from the system). For example:

  /usr/adm/fbackupfiles/indices/usr-usrlib/871128.15:17
  (for Nov 28, 1987 at 3:17 PM)

- Invoke **fbackup** with this path name as its index file name. For example:

```
cd /usr/adm/fbackupfiles
/etc/fbackup -0uc config -g graphs/usr-usrlib\
    -I indices/usr-usrlib/871128.15:17\
        -f /dev/rmt/0h
```

When the session completes successfully, the index is automatically placed in the proper location.

Note that **fbackup** should be piped to **tcio** when backing up to a CS/80 cartridge tape device see *tcio*(1)). The following example copies the entire contents of directory **/usr** to a cartridge tape:

```
/etc/fbackup i /usr -f - | tcio -oe /dev/rct/c0d1s2
```

WARNINGS

Starting with HP-UX Release 8.0, **fbackup** does not back up network special files because RFA networking is obsolete. A warning message is issued if a network special file is encountered in the backup graph and the file is skipped.

The use of **fbackup** for backing up NFS mounted file systems is not guaranteed to work as expected if the backup is done as a privileged user. This is due to the manner in which NFS handles privileged-user access by mapping user **root** and uid **0** to user **nobody**, usually uid **-2**, thus disallowing root privileges on the remote system to a root user on the local system.

The utility set comprised of **fbackup** and **frecover** was originally designed for use on systems equipped with not more than one gigabyte of total file system storage. Although the utilities have no programming limitations that restrict users to this size, complete backups and recoveries of substantially larger systems can cause a large amount system activity due to the amount of virtual memory (swap space) used to store the indices. Users who want to use these utilities, but are noticing poor system-wide performance due to the size of the backup, are encouraged to backup their systems in multiple smaller sessions, rather than attempting to backup the entire system at one time.

Due to present file-system limitations, files whose inode data, but not their contents, are modified while a backup is in progress might be omitted from the next incremental backup of the same graph. Also, **fbackup** does not reset the inode change times of files to their original value.

fbackup allocates resources that are not returned to the system if it is killed in an ungraceful manner. If it is necessary to kill **fbackup**, send it a **SIGTERM**; not a **SIGKILL**.

For security reasons, configuration files and the **chgvol** and **error** executable files should only be writable by their owners.

If sparse files are backed up without using data compression, a very large amount of media can be consumed.

fbackup does not require special privileges. However, if the user does not have access to a given file, the file is not backed up.

fbackup consists of multiple executable objects, all of which are expected to reside in directory **/etc**.

fbackup creates volumes with a format that makes duplication of volumes by **dd** impossible (see *dd*(1)). Copying an **fbackup** volume created on one media type to another media type does not produce a valid **fbackup** volume on the new media because the formats of volumes on 9-track tape, backup to a file, rewritable optical disks and DDS-format tapes are not identical.

When configuring the parameter **blocksperrecord** (see **-c** option), the record size is limited by the maximum allowed for the tape drive. Common maximum record sizes include 16 1-Kbyte blocks for tape drive models HP 7974 and HP 7978A, 32 blocks for the HP 7978B, 60 blocks for the HP 7980, and 64 blocks for DDS tape drives. Note also that the *blocksize* used in earlier releases (7.0 and before) was 512 bytes,

whereas it is now 1024 bytes. This means that the same value specified in *blocksperrecord* in an earlier release creates blocks twice their earlier size in the current release (i.e., a *blocksperrecord* parameter of 32 would create 16-Kbyte blocks at Release 7.0, but now creates 32-Kbyte blocks). If *blocksperrecord* exceeds the byte count allowed by the tape drive, the tape drive rejects the write, causing an error to be communicated to **fbackup** which **fbackup** interprets as a bad tape. The resulting write error message resembles the following:

```
fbackup (3013): Write error while writing backup at tape block 0.
Diagnostic error from tape 11..... SW_PROBLEM  (printed by driver on console)
fbackup (3102): Attempting to make this volume salvageable.
```
etc.

DEPENDENCIES

NFS

Access control lists of networked files are summarized (as returned in **st_mode** by **stat()**), but not copied to the new file (see *stat*(2)).

Series 800

On NIO-bus machines there can be problems when a CS/80 cartridge tape device is on the same interface card as hard disk devices. If writes longer than 16K bytes are made to the tape device, it is possible to have disk access time-out errors. This happens because the tape device has exclusive access to the bus during write operations. Depending on the system activity, this problem may not be seen. The default write size of **fbackup** is 16 Kbytes.

Series 700/800

fbackup does not support QIC-120, and QIC-150 formats on QIC devices. If **fbackup** is attempted for these formats, **fbackup** fails and the following message is displayed :

```
mt lu X: Write must be a multiple of 512 bytes in QIC 120 or QIC 150
```

AUTHOR

fbackup was developed by HP.

FILES

/usr/adm/fbackupfiles/dates	database of past backups

SEE ALSO

cpio(1), ftio(1), tcio(1), dump(1M), frecover(1M), ftio(1M), restore(1M), rmt(1M), stat(2), cdf(4), acl(5), mt(7).

NAME

fddiinit - initialize FDDI network interface; connect to FDDI network

SYNOPSIS

`/etc/fddiinit` [`-l` *download_file*] [`-s`] *device_file*

DESCRIPTION

`fddiinit`:

- Downloads firmware to the FDDI network interface and connects the interface to the FDDI network.
- Must be executed for each interface present on a machine.
- Can also be executed from within the `/etc/netlinkrc` script during network initialization.
- Is also used to reinitialize and reconnect the interface after the interface has been reset. Use the `fddidstop` command to reset the interface (see *fddidstop*(1M)).

Options and Command-Line Arguments

`fddiinit` recognizes the following options and command-line arguments:

`-l` *download_file*	Specifies the firmware download file. See DEPENDENCIES for machine-dependent details.
`-s`	(silent) Suppress the progress message. While `fddiinit` is running, it periodically prints a series of dots on the terminal screen to indicate that the firmware download is in progress.
device_file	Specifies the device special file associated with the FDDI interface. By convention, device special files are kept in the `/dev` directory. Each device file has a name and a device number to uniquely identify the interface. See DEPENDENCIES for a description of how to create device files.

RETURN VALUE

Upon successful completion, `fddiinit` returns 0; otherwise, it returns 1.

ERROR MESSAGES

`fddiinit` fails and the firmware is not downloaded if any of the following conditions are encountered:

- Command used incorrectly – usage message is returned.
- Invalid device file – returns message `Can't open device file`. Check the device file. See DEPENDENCIES for description of how to create device files.
- Invalid download file – returns `Can't open download file` or `Invalid file format`. Contact your HP Customer Support representative.
- Hardware or driver error – download was unsuccessful because of a hardware or firmware problem. Check to ensure that hardware is correctly connected. If the download is still unsuccessful, replace the card with a known-good unit if one is available, and retry the command. Otherwise, contact your HP customer support representative.

DEPENDENCIES

Series 700 EISA FDDI:

Each device file has a name and a device number to uniquely identify the interface. To create the EISA FDDI device file manually (instead of through SAM), specify the applicable major and minor numbers in the HP-UX `/etc/mknod` command. EISA FDDI device files have the following major and minor numbers:

Major	Minor	EISA slot
49	0x410000	1
49	0x420000	2
49	0x430000	3
49	0x440000	4

The following example uses `/etc/mknod` to create the EISA FDDI device special file `/dev/lan1` on the first EISA slot:

```
/etc/mknod /dev/lan1 c 49 0x410000
```

If the FDDI interface card is configured using **sam** (see *sam*(1M)), **sam** creates the device file automatically and the name corresponds to the network interface name and unit. For example, device files **/dev/lan1** and **/dev/lan2** are for network interfaces **lan1** and **lan2** respectively. You can also use the **/etc/lanscan** command to display information about the network interfaces on the system.

fddiinit requires a download file only for the EISA FDDI card. The default download file is **/etc/fddi/fddi_dnld**.

Series 800:

Device files for HPPB FDDI are created automatically by **/etc/insf** (see *insf*(1M)) when the system is rebooted after installing the HPPB FDDI driver and adapter card. The device file name is of the form **/dev/lan***X* where *X* >= 0.

The major number for HPPB FDDI device files is 49. The minor number containing the logical unit (lu number) is assigned based on the position of the HPPB FDDI card in the HPPB backplane relative to other LAN cards. Each LAN card has a unique minor device number.

To determine the device special file corresponding to a particular FDDI adapter, first use the **/etc/lanscan** command (see *lanscan*(1M)) to obtain the lu number that matches the hardware path of that adapter. Then use the **/etc/lssf** command (see *lssf*(1M)) on those files in the **/dev** directory that have a major number of **49** to find a file that has a matching lu number.

mksf (see *mksf*(1M)) can be used to manually create a device file for the HPPB FDDI interface.

The default download file is **/etc/fddi/fddi_dnld**. This download file is used for the HPPB FDDI card.

AUTHOR

fddiinit was developed by HP.

FILES

/etc/fddi/fddi_dnld default FDDI download file.

SEE ALSO

fddistop(1M), fddistat(1M), fddinet(1M), mknod(1M), lanscan(1M).

NAME

fddinet - display logical FDDI ring map information

SYNOPSIS

`/usr/bin/fddinet` [`-n`] [`-d` *station_address*] *device_file*

DESCRIPTION

`fddinet` displays logical connection information for the reachable nodes connected to the same FDDI ring.

Options and Command-Line Arguments

`fddinet` recognizes the following options and command-line arguments:

`-n` Use FDDI native form when displaying address information. The default is the canonical form.

`-d` *station_address*
Specifies the MAC Address of the node that is to be first in the display of the logical ring map. If the `-n` option is used in the command line, the MAC Address is a 12-character, hexadecimal-digit string in FDDI native form; otherwise, the default canonical form is used. It can start with or without the usual `0x` prefix. For example, both `0x080009091219` and `080009091219` are valid MAC Addresses.

device_file Device special file associated with the FDDI interface. By convention, device files are kept in the `/dev` directory. Each device file has a name and a device number to uniquely identify the interface. See the DEPENDENCIES section of *fddiinit*(1M) for a description of how to create device files.

RETURN VALUE

Upon successful completion, `fddinet` returns 0; otherwise it returns 1.

ERROR MESSAGES

`fddinet` fails if any of the following conditions is encountered:

- Command used incorrectly - Usage message is returned.
- Invalid device file - returns `Can't open device file`. Check the device file. See DEPENDENCIES section of *fddiinit*(1M) for a description of how to create device files.
- Hardware or driver error - hardware failed to respond to the request. Ensure that the hardware is correctly connected, then use `fddiinit` to reinitialize the interface if necessary (see *fddiinit*(1M)). If the same failure happens after interface reinitialization, replace the interface with a known-good unit, if one is available, and retry the command. Otherwise, contact your HP Customer Support representative.

AUTHOR

`fddinet` was developed by HP.

SEE ALSO

fddiinit(1M), fddistop(1M), fddistat(1M), mknod(1M).

EXAMPLE OUTPUT

MAC_Address	Node_Type	UNA	Topology
0x080009091319	SAS Station(1 MAC)	0x080009091329	Wrapped
0x080009091329	SAS Station(1 MAC)	0x08006A0D0225	Wrapped
0x08006A0D0225	Concentrator(6 Port)	0x08000909133F	Rooted
0x08000909133F	SAS Station(1 MAC)	0x080009091319	Wrapped

Fields are defined as follows:

MAC_Address Specifies the 48-bit MAC Address of the node in hexadecimal format. The default is canonical form. FDDI native form is used if the `-n` option appears in the command line.

Node_Type Specifies whether the node is a Single Attachment Station (SAS), Dual Attachment Station (DAS), or Concentrator. SAS and DAS station types include the MAC count displayed inside parentheses after the node type; concentrator station types include the number of master ports inside parentheses after the node type.

UNA Specifies the MAC Address of the upstream neighbor in hexadecimal format. The default is canonical form. FDDI native form is used if the `-n` option appears in the command line.

Topology Displays the topology of the station. Possible values are:

`Wrapped`	Set when the station's attachment state is `Wrap_A`, `Wrap_B`, `Wrap_S`, or `Wrap_AB`.
`Unrooted`	Set when a concentrator has no active A, B or S Port.
`Twisted A-A`	Set when an A-A connection is detected in the station.
`Twisted B-B`	Set when a B-B connection is detected in the station.
`Rooted`	Set when the station does not have an A or B or S Port active in tree mode.
`SRF`	Set if the station supports the Status Report (SRF) protocol.

NAME

fddisetup - initialize and connect all system FDDI network interfaces

SYNOPSIS

`/etc/fddisetup`

DESCRIPTION

`fddisetup`:

- Scans the kernel I/O system data structures for all FDDI interface cards installed on the system. It invokes `fddiinit` with default parameters for every FDDI interface card found (see *fddiinit*(1M)).
- The `fddisetup` command must be present in the `/etc/netlinkrc` file and the `/etc/powerfail` file. The entry in the `/etc/netlinkrc` file must be present before any `ifconfig` commands. It is invoked at system start-up to download all FDDI cards in the system before IP addresses are assigned. The entry in the `/etc/powerfail`(1M) file must be present after the `/etc/dasetup` entry. It is invoked when the system recovers from a power-failure condition to reinitialize the FDDI interface cards.
- It can also be invoked manually. However, this is not recommended as it reinitializes all FDDI interface cards present on the system. When it is necessary to reinitialize and reconnect a specific interface, use `fddiinit` interactively.

Command-Line Arguments

`fddisetup` does not support any options or arguments.

RETURN VALUE

Upon successful completion, `fddisetup` returns 0; otherwise, it returns 1.

ERRORS

If `fddisetup` fails, the firmware may be downloaded on some FDDI cards on the system and not on others. If this happens, use `lanscan` to determine which FDDI interface cards have been downloaded properly and which ones were not (see *lanscan*(1M)). `lanscan` shows an `UP` hardware state for cards that have been downloaded properly. Use `fddiinit` interactively to manually download FDDI interface cards.

The error message `fddisetup: Can't read I/O configuration` indicates an access permissions problem. Log in as super-user and try the command again.

`fddisetup` can also produce error messages returned by the `fddiinit` command (see *fddiinit*(1M)).

AUTHOR

`fddisetup` was developed by HP.

SEE ALSO

fddistop(1M), fddistat(1M), fddinet(1M), mknod(1M), lanscan(1M).

NAME

fddistat - show FDDI interface status

SYNOPSIS

`/usr/bin/fddistat` [`-n`] *device_file*

DESCRIPTION

`fddistat` displays information about the status of the FDDI interface.

Options and Command-Line Arguments

`fddistat` recognizes the following options and command-line arguments:

`-n` Use FDDI native form when displaying address information. The default is canonical form.

device_file Specifies the device special file associated with the FDDI interface. By convention, device files are kept in the `/dev` directory. Each device file has a name and a device number to uniquely identify the interface. See DEPENDENCIES section of *fddiinit*(1M) for a description of how to create device files.

RETURN VALUE

Upon successful completion, `fddistat` returns 0; otherwise it returns 1.

ERROR MESSAGES

`fddistat` fails if any of the following conditions is encountered:

- Command used incorrectly - Usage message is returned.
- Invalid device file - returns `Can't open device file .` Check the device file. See DEPENDENCIES section of *fddiinit*(1M) for a description of how to create device files.
- Hardware or driver error - hardware failed to respond to the request. Ensure that hardware is correctly connected, and use the `fddiinit` command to reinitialize the interface if it is necessary (see *fddiinit*(1M)). If the same failure occurs after the interface is reinitialized, replace the interface with a known-good unit, if available, then retry the command. Otherwise, contact your HP Customer Support representative.

AUTHOR

`fddistat` was developed by HP.

SEE ALSO

fddiinit(1M), fddistop(1M), fddinet(1M), mknod(1M), netstat (1M).

EXAMPLE OUTPUT

```
MAC_Address     0x080009091335
UNA             0x080009091189
RMT             Ring_Op
CF_State        Wrap_S
Frame_Ct        5000
Receive_Ct      3500
Transmit_Ct     4000
Lost_Ct         12
Error_Ct        1
LER_Estimate    10**-15
T_Req (ms)      150
T_Neg (ms)      150
```

Fields are defined as follows:

MAC_Address Medium Access Control (unit) Address. Specifies the 48-bit MAC Address of the node in hexadecimal format. The default is canonical form. FDDI native form is used if the `-n` option is specified in the command line.

UNA Upstream Neighbor's (MAC) Address. Specifies the MAC Address of the upstream neighbor in hexadecimal format. The default is canonical form. FDDI native form is used if the `-n` option appears in the command line.

RMT Ring Management State. Indicates whether the state is: Isolated, Non_Op, Ring_Op, Detect, Non_Op_Dup, Ring_Op_Dup, Directed, or Trace. Refer to the RMT description in the ANSI FDDI/SMT specification for more details.

CF_State (Attachment) Configuration State of the station. Indicates whether the state is: Isolated, Wrap_S, Wrap_A, Wrap_B, Wrap_AB, or Thru. Only the Isolated and the Wrap_S states are valid for single attachment station (SAS). Refer to the CF_State variable description in the ANSI FDDI/SMT specification for more details.

Frame_Ct Frame Count. Specifies the total number of frames received with End Delimiter by the station. This count includes void frames, token frames, beacon frames, claim frames, SMT frames and LLC frames.

Receive_Ct Receive Count. Specifies the total number of SMT or LLC frames successfully received by the station.

Transmit_Ct Transmit Count. Specifies the total number of transmit frames originated by this station.

Lost_Ct Lost Count. Specifies the total number of frames received with format error detected. When the station detects a frame with a format error, it strips the rest of the frame from the ring and replaces it with Idle symbols.

Error_Ct Error Count. Specifies the total number of frames received with the End Delimiter not set (not 'S').

LER_Estimate Link Error Rate Estimate. Specifies the long term average link error rate. It ranges from 10^{-4} to 10^{-15}.

T_Req Token Request Time. Specifies the requested Target Token Rotation Time (TTRT) by the local station in the claim token process. The value of *T_Req* is in milliseconds. Refer to the T_Req value description in the ANSI FDDI/SMT specification for more details.

T_Neg Negotiated Target Token Rotation Time (TTRT). Specifies the target rotation time being used by all the stations on the ring. This value is negotiated during the claim token process. The value of *T_Neg* is in milliseconds. Refer to the *T_Neg* value description in the ANSI FDDI/SMT specification for more details.

NAME

fddistop - stop and reset the FDDI interface

SYNOPSIS

`/etc/fddistop` *device_file*

DESCRIPTION

`fddistop` disconnects the interface from the FDDI ring and resets the firmware to the reset state. Use the `fddiinit` command to reinitialize the interface and reconnect to the FDDI network (see *fddiinit*(1M)).

Command-Line Arguments

`fddistop` recognizes the following command-line argument:

device_file Specifies the device special file for the FDDI interface. By convention, device files are kept in the `/dev` directory. Each device file has a name and a device number to uniquely identify the interface. See DEPENDENCIES section of *fddiinit*(1M) for a description of how to create device files.

RETURN VALUE

Upon successful completion, `fddistop` returns 0; otherwise it returns 1.

ERROR MESSAGES

`fddistop` fails if any of the following conditions are encountered:

- Command used incorrectly – usage message is returned.
- Invalid device file – returns `Can't open device file .` Check the device file. See DEPENDENCIES section of *fddiinit*(1M) for a description of how to create device files.

AUTHOR

`fddistop` was developed by HP.

SEE ALSO

fddiinit(1M), fddistat(1M), fddinet(1M), mknod(1M).

NAME

ff - list file names and statistics for a file system

SYNOPSIS

`/etc/ff` [*options*] *special*

DESCRIPTION

`ff` reads the i-list and directories of the special file *special*, assuming it to be a file system, saving i-node data for files that match the selection criteria. Output consists of the path name for each saved i-node, plus any other file information requested using the print options below. Output fields are positional. The output is produced in i-node order; fields are separated by tabs. The default line produced by `ff` includes the path name and i-number fields. With all options specified, the output fields include path name, i-number, size, and UID.

The *num* parameter in the options descriptions is a decimal number, where *+num* means more than *num*, *-num* means less than *num*, and *num* means exactly *num*. A day is defined as a 24-hour period.

`ff` lists only a single path name out of many possible ones for an i-node with more than one link, unless you specify the `-l` option. With `-l`, `ff` applies no selection criteria to the names listed. All possible names for every linked file on the file system are included in the output. On very large file systems, memory may run out before ff does.

Options

`ff` recognizes the following options and command-line arguments:

`-a` *num*	Selects if the i-node has been accessed in *num* days.
`-c` *num*	Selects if the i-node has been changed in *num* days.
`-i` *inode*	Generates names for only those i-nodes specified in the i-node list.
`-I`	Does not display the i-node number after each path name.
`-l`	Generates a list of all path names for files with more than one link.
`-m` *num*	Selects if the file associated with the i-node has been modified in *num* days.
`-n` *file*	Selects if the file associated with the i-node has been modified more recently than the specified *file*.
`-p` *prefix*	Adds the specified *prefix* to each path name. The default prefix is `.` (dot).
`-s`	Writes the file size, in bytes, after each path name.
`-u`	Writes the owner's login name after each path name.

EXAMPLES

List the path names and i-numbers of all files in the file system `/dev/hd1`:

```
ff /dev/hd1
```

Same, but suppress printing of i-numbers:

```
ff -I /dev/hd1
```

List files on the same file system that have been modified recently, displaying the path name, i-number, nad owner's user name (-u option). List only files that have been modified within the last two days (`-m -2` option):

```
ff -m -2 -u /dev/hd1
```

List all files on the same file system, including the path name and i-number of each file, that was last accessed more than 30 days ago (`-a +30`):

```
ff -a +30 /dev/hd1
```

Find all path names associated with i-nodes `451` and *76* (-l option):

```
ff  -l  -i  451,76  /dev/hd1
```

NAME

fingerd - remote user information server

SYNOPSIS

/etc/fingerd [**-r**]

DESCRIPTION

fingerd is the server for the RFC 742 Name/Finger protocol. It provides a network interface to **finger**, which gives a status report of users currently logged in on the system or a detailed report about a specific user (see *finger*(1)). The Internet daemon executes **fingerd** when it receives a service request at the port listed in the services data base for "finger" using "tcp" protocol; see *inetd*(1M) and *services*(4).

To start **fingerd** from **inetd**, the configuration file **/etc/inetd.conf** must contain an entry as follows:

```
finger stream tcp nowait bin /etc/fingerd fingerd
```

Once a remote host is connected, **fingerd** reads a single "command line" terminated by a carriage-return and line-feed. It uses this command line as the arguments to an invocation of **finger**. **fingerd** sends the output of **finger** to the remote host and closes the connection.

If the command line is null (contains only a carriage-return and line-feed pair), **finger** returns a report that lists all users logged in on the system at that moment.

If a user name is specified on the command line (for example, *user*<CR><LF>), the response lists more extended information for only that particular user, whether logged in or not. See *finger*(1) for the details of this extended information.

If **fingerd** is run with the **-r** option, it allows remote user names on the command line (for example, user@host<CR><LF>). Otherwise, if the command line contains a remote user name, **fingerd** prints the error message **Remote finger not allowed** and closes the connection.

AUTHOR

fingerd was developed by the University of California, Berkeley and HP.

SEE ALSO

finger(1), inetd(1m), services(4),

RFC 742 for the Name/Finger protocol.

NAME

fpkg - file packaging utility for use with *update*(1m)

SYNOPSIS

fpkg [**-m** *media_type*] [**-d** *destination_directory*] [**-a** *archive_file*] [**-s** *device_size*] [**-V** *media_format_version*] [**-S** *machine_series*] [**-c** *comment_string*] [**-L** *log_file*] [**-hvM**] *Product_Specification_File*

fpkg [**-L** *log_file*] [**-v**] **-r** *media_directory* > *Product_Specification_File*

DESCRIPTION

fpkg is used to package a collection of files and produce media in a format usable by **update** (see *update*(1M)). The media produced can be either tape archive format (see *tar*(1)) or in a format usable by **netdistd** (see *netdistd*(1M)). **fpkg** cannot be used to make CD-ROM media. When making network media **fpkg** must be executed by user **root** (super-user) so that file modes can be correctly set under the **netdist** directory. Making tape media does not require root privileges unless source files cannot otherwise can be read. For tape media, file modes are set inside the archive and no special privileges are required.

fpkg packages files according to the specifications in the user-supplied *Product_Specification_File* (PSF), unless the **-r** option is specified. With the **-r** option, **fpkg** examines the given *media_directory*, and writes a PSF to the standard output. The PSF that is produced can be used as input to a second invocation of **fpkg** in order to repackage *media_directory*. *media_directory* can be a **netdistd** directory, or a mounted CD-ROM (See **-r** description for details).

Options

fpkg recognizes the following options:

-m *media_type* Valid *media_type* is either **network** or **tape**. Setting *media_type* to **network** causes **fpkg** to load the package into the *destination_directory* (**-d** option) in a format suitable for use by **netdistd**. Setting *media_type* to **tape** causes **fpkg** to write the package to the specified *archive_file* (**-a** option) in **tar** format which is directly usable by **update**. Specifying the **-m** option is redundant information and not necessary if the **-a** or **-d** option has been specified (these options imply the media type).

The default *media_type* is **network**.

-d *destination_directory*
Specifies the directory in which **fpkg** creates the package in a format suitable for use by **netdistd**. This option implies a *media_type* (**-m** option) of **network**. **fpkg** must be executed by root (super-user) when making network media.

The default *destination_directory* is **/netdist**.

-a *archive_file*
Specifies the output device file (or regular disk file) for **fpkg** to write the tape archive. The output device (or file) can then be used directly by **update**. The output is also readable by **tar** (although only the table-of-contents option is of real use). This option implies a *media_type* (**-m** option) of type **tape**. An *archive_file* argument of **-** causes fpkg to write the resulting archive to standard out. This can be useful when writing to a device on a remote host (see EXAMPLES).

The default *archive_file* is **/dev/rmt/0m**.

-s *device_size* Specifies the *archive_file* capacity in Mbytes (where 1 Mbyte = 1024 × 1024 bytes). Used only if the *media_type* (**-m** or **-a** option) is **tape**. The size information is used to determine how much of the package can fit on one tape volume. This information is necessary when the package spans more than one tape volume. For some devices **fpkg** can automatically determine the capacity. For those devices the default sizes are:

Cartridge tape:	63 Mbytes
9-track tapes:	40 Mbytes
DDS tapes:	1330 Mbytes

Regular disk file: size of free file system space.

-V *media_format_version*
Specifies the format of media produced by **fpkg**. *media_format_version* determines which versions of **update** are able to read the media produced. **update** is able to read media marked with a *media_format_version* older than the time of its release, but not newer. Acceptable values for *media_format_version* are: **A.B7.00**, **A.B8.00**, and **A.B8.05**. If *media_format_version* does not match one of these, it is rounded down to one of the accepted values and a notice message is produced. Only **A.B8.05** media fully supports the S700.

The default *media_format_version* is **A.B8.00**.

-S *machine_series*
Specifies which series of machines can read the media produced. Acceptable values are: **300** (implies also Series 400), **700**, and **800**. For tape media it is possible to allow all series to read the media, in which case the **-S** option should be omitted and the PSF should not set any architecture specifiers (**sys** or **is** keywords or **H** or **M** flags to the **ff** keyword). For network media, this information is needed because an element of the **netdistd** directory structure contains the series name.

For tape media of version **A.B8.05**, it is possible to specify a mixture of machines that can load the media. In this case, multiple **-S** options can be specified (such as **-S700** or **-S800**).

For network media, **fpkg** uses the **-S** option to determine which subdirectory (300, 700, or 800) under the *destination_directory* to place the package. If multiple **-S** options are given, **fpkg** uses the first one found for determining the netdist subdirectory. Therefore **fpkg** must be executed multiple times, rearranging the **-S** options so that the package can be placed in each of the appropriate subdirectories.

The default *machine_series* is "all series" for tape media. For network media it defaults to the machine series that **fpkg** is executed from. This default value is overridden if the PSF contains any machine-architecture specifiers (**fpkg** issues a notice in this case).

-c *comment-string*
Allows the user to override the default comment string that is placed in the **MAIN.pkg** file used by **netdistd**. The default string is: **Fileset packages for use by update(1M)**.

-L *logfile*
Changes the log file from the default: **/tmp/fpkg.log**, to *logfile*. **fpkg** appends a log of messages, errors, option settings, and other information to this file.

-h
This option tells **fpkg** to ignore files that are symbolic links, and to treat the linked-to file as the file to be placed into the package instead of the link. Without the **-h** option, **fpkg** stores the symbolic link as it appears. **update** then restores the symbolic link when the media is loaded.

-v
Sets verbose mode. **fpkg** prints information that is normally not necessary. This is useful for determining what defaults were chosen and for a step-by-step progress report. Normally **fpkg** issues some status information, notices, and errors.

-M
Setting this option allows **fpkg** to produce media that contains filesets destined for a mixture of architectures. However, until HP-UX Release 9.0, **update** refuses to load media that contains filesets with mixed architecture specifiers. Using this option may cause the media to be loadable only by HP-UX Release 9.0 or later systems. **fpkg** gives a warning if this is the case.

-r *media_directory*
Recreates a PSF that can be used in a second invocation of **fpkg** to repackage the filesets found in *media_directory*. This option is used to transfer filesets from either CD-ROM or netdistd media to tape. *media_directory* is the path name of a mounted CD-ROM, or that of the architecture level of a netdistd directory (such as **/UPDATE_CDROM**, **/netdist/300**, **/netdist/800**, etc). **updist** should be used when transferring media to a netdistd directory (see *updist*(1M)).

fpkg skips any filesets on CD-ROM media that are secured (encrypted). If the **-v** option is given, a notice is given for each secured fileset skipped. If a secured fileset is to be transferred to the tape media, **updist** must be used (with a valid codeword) to first transfer the fileset or filesets to a netdistd directory where they are in unencrypted form. A PSF can then be created from the netdist media and

appended to the deficient PSF created from the secured CD-ROM. If all desired filesets were loaded by `updist`, the PSF created from the netdistd directory should be complete as-is. Use appropriate precautions when duplicating any material that may be protected by copyright and/or license agreements.

`-?`
Any invalid option causes `fpkg` to display a general usage message.

Product Specification File (PSF) format
Update media appears as a three-layer hierarchy. The top layer contains **partitions**. Under each **partition** is a collection of **filesets**. Each **fileset** contains any number of **files**.

The PSF is used to specify the structure of the package and the attributes of the files and filesets contained in the package. It is composed of recognized keywords and lists of file names to be included in the package (one per line). Extra white space is ignored. Comments are recommended, and consist of a # character followed by optional arbitrary text through end-of-line. In general, the pseudo structure of the PSF is:

```
        Partition name and description
              Fileset name and description
              Fileset attributes
                    Files in fileset
  Another fileset
  Attributes
    Files in this fileset
  Next Partition name and description
    Next Fileset
    Attributes
          Files
  etc, etc, ...
```

Valid Keywords.
The PSF is made up of a list of keywords usually followed by an argument of some kind. For example:

```
pn OS-CORE
```

defines the next partition which is to be named `OS-CORE`. The following table is a list of recognized keywords. Most keywords have a short (abbreviated) form and an unabbreviated, more readable form; the two forms can be used interchangeably:

Keyword	Unabbreviated Form	Argument(s)	Required?
`pn`	`partition_name`	string	Recommended
`pd`	`partition_description`	string	Recommended
`fn`	`fileset_name`	string	Yes
`fd`	`fileset_description`	string	Recommended
`ff`	`fileset_flags`	characters	No
`is`	`instruction_set`	instruction-id	No
`sys`	`system_architecture_type`	Series-list	No
`dep`	`fileset_dependency`	string(s)	No
`fv`	`fileset_version`	version-string	No
`ffperm`	`fileset_file_permission`	owner group mode	No
`fdperm`	`fileset_directory_permission`	owner group mode	No
`customize`		filename	No
`decustomize`		filename	No
`copyright`		filename	No
`CDFinfo`		filename	No
`systemfile`		filename	No
`media_order`		number	No
`media_format`		format-version	No
`pr`	`pseudo_root`	path[=path]	No
`F`	`Files`	*, or none	Yes

Keyword Descriptions.
The recognized keywords are described below:

pn | partition_name
Example: **pn OS-CORE**
Indicates the partition name to which the filesets that follow this keyword belong. This keyword must precede the partition description and any fileset specifiers within the partition. Maximum length is 14 characters; anything beyond 14 is ignored.

pd | partition_description
Example: **pd "The base product"**
Gives the current partition a short description string. Maximum length is 32 characters; anything beyond 32 is ignored.

fn | fileset_name
Example: **fn UX-CORE**
Always necessary and is used to mark and name the beginning of a new fileset. Must precede any other keywords associated with this fileset (except for partition information). Maximum length is 14 characters; anything beyond 14 is ignored. Note that each fileset name must be unique, both on the update media and when loaded onto a system. Care should be taken when naming filesets to avoid fileset name collisions. Look in the **/system** or **/etc/filesets** directory for examples of names already in use on your system (this is not by any means the complete set).

fd | fileset_description
Example: **fd "OS Runtime files"**
Gives the current fileset a short description of what purpose the fileset serves. Maximum length is 32 characters; anything beyond 32 is ignored.

ff | fileset_flags
Example: **ff CD**
Used to assign special attributes to a fileset. Depending on these attributes, **update** treats this fileset differently. The argument to this keyword is a list of characters; each character has a special meaning (order is not important). Possible character flags are:

- **B** Causes **update** to rebuild the kernel and reboot after the fileset is loaded and after its customize script is run. All filesets marked with a **B** flag are loaded before the kernel is rebuilt.
- **C** Indicates that this fileset cannot be loaded under a user-specified directory; it must be loaded relative to **/**.
- **Y** Indicates to **update** that it should run **sysrm** or **rmfn** to remove any existing fileset having the same name, prior to reloading. This can slow the **update** process considerably and is not normally done. It is best to remove specific unwanted files in the customize script.
- **D** Used to indicate that the fileset's customize script should not be run until after all filesets have been loaded. This is the default action for filesets loaded with 8.05-or-later **update**. Thus this flag is obsolete (but can still be specified) for loading on 8.05 or later systems. This flag is not compatible with the **B** flag.
- **S** Used only if this media is later transferred to a CD-ROM through HP's integration and manufacturing process. When this flag is set, the fileset is encrypted upon transfer to the CD. When encrypted, the fileset cannot be loaded without first obtaining a codeword (password). Note that **fpkg** cannot make CD-ROM media.

The architecture-specific flags **H** and **M** are used on A.B7.00 and A.B8.00 version media. For A.B8.05 media, use the **sys** and **is** keywords instead. The **H** and **M** flags may be used to specify the types of machines that can load this fileset. These flags can be left off to indicate that the fileset is loadable by all series machines. Or they can be left off and later specified by using the **-S** command line option, in which case **fpkg** automatically supplies these flags (if the media version is A.B7.00 or A.B8.00 — if the media version is A.B8.05, it uses the appropriate **sys** and **is** keywords). All filesets on the media must have the same architecture-specific flags (unless the **-M** option is used). It is recommended that neither the **H** nor **M** flag be specified and that the **-S** option be used, causing **fpkg** to supply the appropriate flags.

H For A.B7.00 and A.B8.00 version media, this flag is used to indicate that this fileset is loadable only onto HP-PA-RISC architecture machines, namely Series 800 and Series 700. The **H** and **M** flags cannot be used simultaneously.

M Used to specify the architecture type, like the **H** flag, but instead indicates compatibility with MC-680x0 series machines; namely Series 300 and Series 400. Cannot be used simultaneously with the **H** flag.

sys | system_architecture_type
Example: **sys S700,S800**
Can be used to specify which series of systems are allowed to load this fileset. This keyword is only valid with A.B8.05-version media. Valid system types are **S300**, **S400** (translated to be S300), **S600** (translated to be S800), **S700**, **S800**, or * (meaning any). One or more series specifiers can be used, separated by commas. If this keyword is used, the **is** keyword must also be specified. This keyword cannot be used in conjunction with the **H** or **M** flag to the **ff** keyword. It is recommended that neither the **sys** or **is** keywords be used, and the **-S** command-line option be used to allow **fpkg** to automatically generate this keyword (as appropriate). Using the **sys** and **is** to specify the architecture overrides the value set with the **-S** option. This keyword is used primarily when some filesets are for a different architecture.

is | instruction_set
Example: **is PA_RISC_1_0**
Can be used to specify the instruction set of systems allowed to load this fileset. This keyword is only valid with A.B8.05 version media. Valid instruction-set identifiers are: **MC68020** (for Series 300 and Series 400 machines), **PA_RISC_1_0** (for Series 700 and Series 800), **PA_RISC_1_1** (for Series 700 only), or *, indicating that any instruction-set machine can load this fileset. If this keyword is used, the **sys** keyword must also be specified. This keyword cannot be used in conjunction with the **H** or **M** flag to the **ff** keyword. It is recommended that neither the **sys** nor **is** keywords be used and the **-S** command line option be used to allow **fpkg** to automatically generate this keyword (as appropriate). Specifying the architecture with the **sys** and **is** keywords overrides the value set with the **-S** option. This keyword is mainly used when not all filesets are for the same architecture.

dep | fileset_dependency
Usage: **dep** *fileset* [*version*]
Example: **dep UX-CORE A.B8.07.0A**
Can be used to specify any fileset (and fileset version) that must be loaded before, or along with this fileset for the product to function properly. This keyword can be used multiple times for the same fileset if it depends on multiple filesets. **fpkg** gives a warning if the fileset depended on is not contained in the same package. This is because **update** cannot enforce this dependency if it is not on the same media; it can only give a warning during loading. The optional version string is used to specify what version of another fileset this fileset depends on. This version feature is not supported when making A.B7.00 media (**fpkg** gives a warning in this case). See the **fv** keyword for more details. Fileset name must not exceed 14 characters; version is truncated at 11 characters.

fv | fileset_version
Example: **fv A.B8.07.0A**
Sets the version string for the current fileset. The version string is used by **update** to prevent the accidental loading of a fileset that is older than what currently resides on the system. **update** gives a warning if the fileset version being loaded is lower than the version of the same fileset already loaded. The version string is also used in calculating fileset dependencies (refer to the **dep** keyword). Giving a fileset a *version* allows other filesets to depend on a particular version of this fileset. For example, if this fileset is loaded onto a system and has a fileset version of A.B8.07.0A (as in the above example), and later another fileset is loaded that has a dependency on version A.B8.05.0A of this fileset, **update** proceeds with the load because it knows that the system contains a fileset equal or greater than the version required. The concept of giving a fileset a version number was introduced at HP-UX Release 8.0. Therefore when making media for HP-UX 7.0, the fileset version is ignored, and **fpkg** gives a warning (if the media version (**-V**) is A.B7.00). Giving a fileset version A.B7.00 (the default) indicates to **update** that it should not use the version number in its calculations, and it always reloads the fileset if another selected fileset depends on it.

update (and thus **fpkg**), requires that the fileset version be at least A.B7.00; thus a version of A.B6.5 is rejected.

Fileset version strings are a sequence of dot-separated letters and digits. When **update** compares two version strings, it compares each corresponding sub-string between the dots. So a version of B6 is greater than A.B7.00. Version strings are truncated at 11 characters.

ffperm | **fileset_file_permission**
Usage: **ffperm** *owner group mode*
Example: **ffperm root bin 0555**
The **ffperm** or **fileset_file_permission** keyword is used to set the default file permissions on files listed following this keyword up to the next occurrence of the **ffperm** keyword or the beginning of the next fileset (determined by the **fn** keyword). If this keyword is not given for a particular fileset, the default action is that each file inherits the permissions (owner, group and mode) of the source file. This keyword is most useful when a group of files all have the same permissions. To set the permissions on a per-file basis, or to override the default permissions, the **-o**, **-g**, and **-m** file flags can be used (see the **F** keyword for more details). File modes are assumed to be specified in octal, a leading **0** can be used to emphasize this, but is not necessary. To specify a hexadecimal mode, use a **0x** prefix. Modes cannot be specified in decimal. Any field (owner, group, or mode) can be left to take the default action by specifying a * character in its place. Default permissions for directories can be set using the **fdperm** keyword.

fdperm | **fileset_directory_permission**
Usage: **fdperm** *owner group mode*
Example: **fdperm root bin 0777**
The **fdperm** or **fileset_directory_permission** keyword is very similar to the **ffperm** keyword, except that it is used to specify the default permissions for directories. See **ffperm** description for details.

customize
Example: **customize /foo/filename**
Allows a customize script to be placed on the media and associated with the current fileset. This script is executed after the fileset has been successfully loaded. The customize script is executed with the current working directory set to where the fileset is loaded (usually **/**, but can be relocated as specified by the user, if the fileset flag **ff C** is not specified). The customize script is passed one argument, which is either **HP-MC68020** or **HP-PA** depending on which type of machine the fileset is loaded for (this is useful when loading on a mixed-architecture cluster).

decustomize
Example: **decustomize /users/joe/file**
Allows a decustomize script to be placed on the media and associated with the current fileset. This script is executed when the fileset is removed using **rmfn** (see *rmfn*(1M)). It is important to know that the decustomize script is executed twice. The first time **rmfn** runs the script is just to check if the fileset is removable; the second time, it is called just prior to the removal of all files loaded with this fileset. It is during the second invocation that actions related to file removal should take place (such as killing processes, etc). The first invocation of the script is given two arguments: the machine architecture (**HP-MC68020** or **HP-PA**), and the word **check** (meaning don't do anything yet, just checking). The second invocation of the script is given just one argument (the machine architecture). The script should exit with a return code of 0 if no problems are encountered, and with the value 1 if an error occurred. The first invocation of the script is the only chance it has to stop the removal process (by exiting with a value of 1).

copyright
Example: **copyright /build/thecopyright**
Used to place a file on the system called: **/system/***fileset-name***/copyright.** This is where most HP applications place copyright information about the product contained in that fileset.

CDFinfo
Example: **CDFinfo /build/cluster.info**
Allows a CDFinfo file to be placed on the media and associated with the current fileset (see *cdfinfo*(4)). The CDFinfo file contains rules that **update** uses when loading the fileset onto a clustered system. These rules specify which files should be loaded as context-dependent files (or CDFs). The rules in this file also apply to **sam** when a system is changed into a cluster server, or when adding a cnode (see *sam*(1M)). A CDFinfo file is not necessary if the application is not supported on HP-UX clusters, or if all the files are system-independent (i.e., can be shared by all systems in a cluster).

systemfile
Example: **systemfile /build/routines**
Used if a file needs to be loaded in the **/system/***fileset-name* directory but has no specific keyword to place it there (i.e., is not a customize, decustomize, copyright, or CDFinfo file). The file is loaded under the fileset directory associated with the current fileset, and is named the same as the basename of the source file.

Do not place files called **index** in this directory; that name is created by **fpkg** and used by **update**, and other utilities. Also if the filesets are to be loaded into a system running 8.0 HP-UX, **update** removes the obsoleted files called **revlist**, **pif**, and **customize,old**. It is best to avoid using system files of these names.

media_order
Example: **media_order 2**
Range: 1-10
Used to control the order in which filesets are written to **tape** media. All filesets with **media_order 1** are processed first, then those with **media_order 2**, etc. However, all filesets that are marked with the **ff B** flag are placed on the media first (with **media_order** used to sort among these **B** flagged filesets). This is because **update** loads all **B** marked filesets first so that the new kernel can be built.

Those filesets with the same **media_order** are placed on the media as they appear in the PSF (this is the default case, all filesets initially having **media_order 1**).

media_format
Example: **media_format A.B8.00**
Can be used to specify the media format version from within the PSF. This value must agree with any *media_format_version* supplied with the **-V** option.

pr | pseudo_root
Usage: **pr** *source-directory*[=*destination-directory*]
Example 1: **pr /users/joe/build**
Example 2: **pr /users/joe/build=/usr/bin**
The **pr** or **pseudo_root** keyword specifies where the source files are to be found on the system, and optionally where those files should be placed when loaded by **update**. The usage shown in example 1 causes **fpkg** to look for the source files in the directory **/users/joe/build**. Any files (not beginning with **/**) specified with the **F** keyword have their path prefixed with the path **/users/joe/build** and included in the current fileset. If the **F *** keyword is used, everything under the directory **/users/joe/build** are included in the current fileset.

The usage in example 2 also causes **fpkg** to look for files in the directory **/users/joe/build**, but the files have the path **/users/joe/build** replaced with the path **/usr/bin** when they are stored on the media. This is very useful if the directory that holds the source files is different than where they should be when loaded by **update**.

F | Files
Usage: **F**[*]
Example 1:

```
F
file1
file2
etc...
```

Example 2: **F ***

Used to specify the files to be included in the current fileset. In the usage shown in example 1, the keyword is used to show that all the file names that follow this line are to be included in the current fileset. All following lines that do not match a reserved **fpkg** keyword are assumed to be file names. If a filename conflicts with a reserved keyword, the filename can be modified by either using its full path, or by inserting **./** in front of the name.

The usage shown in example 2 requires that the **pr** (**pseudo_root**) keyword be specified prior to this keyword. In this usage, the keyword says to start in the *source-directory* specified by the **pr** keyword and recursively include all files and directories found under that directory.

The file names that can follow the **F** keyword can have the following format:

sourcefile [*destination*] [-o *owner*] [-g *group*] [-m *mode*]

For example:

```
#specify a single file
sourcefile

# specify where to get the file and
# what to name it on the media
sourcefile destination

# specify a file, and override permissions
sourcefile -o root -g bin -m 0755
```

EXAMPLES

The examples below show how to use fpkg keywords to construct a useful PSF. They assume that certain files and directory structures already exist on the system where **fpkg** is being executed.

Example PSF:

```
# This fileset uses some of fpkg's more advanced
# features, and has multiple partitions and filesets

# Start of DATABASE partition
pn DATABASE
pd "The Database"

    fn DBASE-RUN
    fd "The database application"
        # set flag to make update load files under '/'
            ff C
            customize /build/scripts/customize-DBASE
            decustomize /build/scripts/decustomize-DBASE
            CDFinfo /build/scripts/CDFinfo-DBASE
            copyright /build/misc/rights
            fv  A.B8.07.1A
        # This fileset depends on 8.07 version of C-MIN
            dep C-MIN A.B8.07.1A
        #
        # The DBASE-RUN product contains everything in
        # /build/dbase/bin, and is loaded on the users system
        # under /usr/bin. These are all executables so set the
        # fileset permissions as such.
        #
        # set default permissions for files and directories
            ffperm bin bin 0655
            fdperm bin bin 0555
        # specify the source and dest directories
            pr /build/dbase/bin=/usr/bin

        # load all files from bin directory
            F *

        # Now add the support files. Set permissions one-by-one
            pr /build=/usr
            F
        # set permissions of directory
            lib -o bin -g other -m 555
        # Load these files from /build/lib to /usr/lib
            lib/dictionary -o root -g bin -m 0444
            lib/library -o root -o bin -m 644

        # Now add some misc files.
            ffperm bin bin 666 # set default perms for files
```

```
        pr /build/misc=/usr/local/misc
        F
        file1
        file2
        ffperm bin bin 555 # set new default perms
        F
        file3
        file4
    # Start the DBASE-DOC fileset now
    fn DBASE-DOC
    fd "Documentation for DBASE"
        copyright /build/misc/rights
        # manuals in correct place on this machine
        pr /usr/man/man1
        # include all manuals as they appear.
        F *
    # Now start another partition, DBEXAMPLES
    pn DBEXAMPLES
    pd "Database examples"
        fn DBASE-EXAMPLE
        fd "Example database's"
            # specify directory permissions
            fdperm bin bin 555
            pr /build/examples=/usr/local/examples
            F
            # override permissions on each file
            # and rename them.
            example1 good -o bin -g bin -m 644
            example2 bad  -o bin -b bin -m 555
            example3 ugly -o bin -b bin -m 444
```

Common usages of **fpkg** are shown below:

Create netdistd media for S300 machines (using the Product Specification File **/tmp/psf**).

```
fpkg -d /netdist/rel1.0 -S300 -v /tmp/psf
```

Create a S800 tape from a existing netdistd directory (2 steps).

```
fpkg -r /netdist/800 > /tmp/psf
fpkg -a /dev/rmt/0m -S800 -v /tmp/psf
```

Create a tape using a device on a remote host (tape size must be specified). It is important to set the output block size with **obs** (as opposed to **bs** which sets the input block size as well).

```
fpkg -a - -s1330 /tmp/psf | remsh host dd obs=10k of=/dev/rmt/0m
```

Create a tape image, then transfer the image to DDS and cartridge tapes, respectively.

```
fpkg -a /tmp/update.image /tmp/psf
dd if=/tmp/update.image of=/dev/rmt/0m bs=10k
cat /tmp/update.image | tcio -o -Z -vV -S 8 /dev/update.src
```

SEE ALSO

update(1M), update(4), netdistd(1M), rmfn(1M), CDFinfo(4), sam(1M), tar(1).
Creating Product Packages for HP-UX manual.

NAME

frecover - selectively recover files

SYNOPSIS

/etc/frecover -r [**-hmosvyAFNOX**] [**-c** *config*] [**-f** *device*] [**-S** *skip*]

/etc/frecover -R *path* [**-f** *device*]

/etc/frecover -x [**-hmosvyAFNOX**] [**-c** *config*] [**-e** *path*] [**-f** *device*] [**-g** *graph*] [**-i** *path*] [**-S** *skip*]

/etc/frecover -I *path* [**-vy**] [**-f** *device*] [**-c** *config*]

/etc/frecover -V *path* [**-vy**] [**-f** *device*] [**-c** *config*]

DESCRIPTION

frecover reads media written by the *fbackup*(1M) command. Its actions are controlled by the selected function **-r**, **-R**, **-x**, **-V**, or **-I**.

The function performed by **frecover** is specified by one of the following letters:

-r The backup media is read and the contents are loaded into the directories from which they were backed up. This option should only be used to recover a complete backup onto a clear directory or to recover an incremental backup after a full level-zero recovery (see *fbackup*(1M)). This is the default behavior.

-x The files identified by the **-i**, **-e**, and **-g** options (see below) are extracted or not extracted from the backup media. If a file to be extracted matches a directory whose contents have been written to the backup media, and the **-h** option is not specified, the directory is recursively extracted. The owner, modification time, and access control list (including optional entries, unless the **-A** option is specified) are recovered. If no file argument is given (including an empty graph file), all files on the backup media are extracted, unless the **-h** option is specified.

-I *path* The index on the current volume is extracted from the backup media and is written to *path*.

-V *path* The volume header on the current volume is extracted from the backup media and is written to *path*. The following fields from the header are extracted in the format *label*:*value* with one pair per line.

Magic Field On a valid **fbackup** media it contains the value **FBACKUP LABEL**.

Machine Identification This field contains the result of **uname -m**.

System Identification This field contains the result of **uname -s**.

Release Identification This field contains the result of **uname -r**.

Node Identification This field contains the result of **uname -n**.

User Identification This field contains the result of *cuserid*(3S).

Record Size This field contains the maximum length in bytes of a data record.

Time This field contains the time **fbackup** was started.

Media Use This field contains the number of times the media has been used for backup.

Volume Number This field contains a # character followed by 3 digits, and identifies the current volume in the backup.

Checkpoint Frequency This field contains the frequency of backup-data-record checkpointing.

Fast Search Mark Frequency This field contains the number of files between fast search marks for backups made with DDS tape drives.

Index Size This field contains the size of the index.

Backup Identification Tag This field is composed of 2 items: the process ID (pid),

and the start time of that process.

`Language` This field contains the language used to make the backup.

`-R` *path*
An interrupted full recovery can be continued using this option. `frecover` uses the information in file *path* to continue the recovery from where it was interrupted. The only command line option used by `frecover` with this option is `-f`. The values in *path* override all other options to `frecover`. Note also that only full recoveries are restarted with this option, because no history of include or exclude lists is stored in the restart file. If a partial recovery (i.e., using the `-x` option) is interrupted then restarted with this option, `frecover` continues recovering where the partial recovery left off, but restores all files on the backup media beyond this point.

The following characters can be used in addition to the letter that selects the desired function:

`-c` *config* *config* specifies the name of a configuration file to be used to alter the behavior of `frecover`. The configuration file allows the user to specify the action to be taken on all errors, the maximum number of attempts at resynchronizing on media errors (`-S` option), and changing media volumes. Each entry of a configuration file consists of an action identifier followed by a separator followed by the specified action. Valid action identifiers are `error`, `chgvol`, and `sync`. Separators can be either tabs or spaces. In the following sample configuration file, each time an error is encountered, the script `/usr/adm/fbackupfiles/frecovererror` is executed. Each time the backup media is to be changed, the script `/usr/adm/fbackupfiles/frecoverchgvol` is executed. The maximum number of resynchronization attempts is five.

```
error  /usr/adm/fbackupfiles/frecovererror
chgvol /usr/adm/fbackupfiles/frecoverchgvol
sync 5
```

`-e` *path*
path is interpreted as a graph to be excluded from the recovery. There is no limit on how many times the `-e` option can be specified.

`-f` *device*
device identifies the backup device to be used instead of the default `/dev/rmt/0m`. If *device* is `-`, `frecover` reads from standard input. Thus *fbackup*(1M) and `frecover` can be used in a pipeline to backup and recover a file system as follows:

```
fbackup -i /usr -f - | (cd /mnt; frecover -Xrf -)
```

If more than one output file is specified, `frecover` uses each one successively and then repeats in a cyclical pattern. Patterns can be used in the device name in a way similar to file name expansion as done by *sh*(1). The expansion of the pattern results in all matching names being in the list of devices used. A device on the remote machine can be specified in the form *machine*`:`*device*. `frecover` creates a server, `/etc/rmt`, on the remote machine to access the tape device. The pattern matching capability does not apply to remote devices. Only half-inch 9-track magnetic tapes or DDS-format tapes can be remote devices. The fast search capability is not used when accessing remote DDS-format devices.

`-g` *graph* *graph* defines a graph file. Graph files are text files and contain the list of file names (graphs) to be recovered or skipped. Files are recovered using the `-i` option; thus if the user wants to recover all of `/usr`, the graph file contains one record:

```
i /usr
```

It is also possible to skip files by using the `-e` option. For instance, if a user wants to recover all of `/usr` except for the subgraph `/usr/lib`, the graph file contains two records:

```
i /usr
e /usr/lib
```

If the graph file is missing, `frecover` exits with an error message. An empty graph file results in recovering all files on the media.

`-h` Extract the actual directory, rather than the files that it references. This prevents hierarchical restoration of complete subtrees from the backup media.

-i *path* *path* is interpreted as a graph to be included in the recovery. There is no limit on how many times the **-i** option can be specified.

-m Print a message each time a file marker is encountered. Using this option, **frecover** prints a message each time either a DDS setmark, a file marker, or a checkpoint record is read. Although useful primarily for troubleshooting, these messages can also be used to reassure the user that the backup is progressing during long, and otherwise silent, periods during the recovery.

-o Recover the file from the backup media irrespective of age. Normally **frecover** does not overwrite an existing file with an older version of the file.

-s Attempt to optimize disk usage by not writing null blocks of data to sparse files.

-v Normally **frecover** works silently. The **-v** (verbose) option causes it to display the file type and name of each file it treats.

-y Automatically answer **yes** to any inquiries.

-A Do not recover any optional entries in access control lists (ACLs). Normally, all access control information, including optional ACL entries, is recovered. This option drops any optional entries and sets the permissions of the recovered file to the permissions of the backed up file. Use this option when recovering files backed up from a system with ACLs on a system for which ACLs are not desired (see *acl*(5)).

-F Recover files without recovering leading directories. For example, this option would be used if a user wants to recover **/usr/bin/vi**, **/bin/sh**, and **/etc/passwd** to a local directory without creating each of the graph structures.

-N (no recovery) Prevent **frecover** from actually recovering any files onto disk, but read the backup as if it was, in fact, recovering the data from the backup, producing the same output that it would on a normal recovery. This option is useful for verifying backup media contents in terms of validity (block checksum errors are reported), and contents (a listing of files can be produced by using the **-N** and **-v** options together). Note that the listing of files produced with the **-N** and **-v** options requires the reading of the entire backup, but is therefore a more accurate reflection of the backup's contents than the index stored at the beginning of the backup (which was created at the start of the backup session, and is not changed during the course of the backup).

-O Use the effective uid and gid for the owner and group of the recovered file instead of the values on the backup media.

-S *skip* **frecover** does not ask whether it should abort the recovery if it gets a media error. It tries to skip the bad block or blocks and continue. Residual or lost data is written to the file named by *skip*. The user can then edit this file and recover otherwise irretrievable data.

-X Recover files relative to the current working directory. Normally **frecover** recovers files to their absolute path name.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which **frecover** expects files to be stored in the backup device and the order in which file names are output by the **-I** option.

LANG determines the language in which messages are displayed.

If **LC_COLLATE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **frecover** behaves as if all internationalization variables are set to "C". See *environ*(5).

WARNINGS

For incremental backups created prior to installing HP-UX Release 8.0, or for recoveries that do not begin with the first volume (such as when, reading tape 3 first), it is possible for the preceding directories to a recoverable file to not be on the media. This can happen, for example, if the directories did not change since

the last full backup. If **frecover** encounters a file on the backup that should be recovered, but it has not recovered the file's parent directories from the backup, it prints a message stating that the recovery will continue with that file, and attempts to create the file's parent directories as needed.

Use of **frecover** does not require special privileges. However, if a user does not have access permission to a given file, the file is not recovered.

Network special files are obsolete. Therefore, **frecover** cannot restore these files. A warning message is issued if an attempt is made to recover a network special file, and the file is skipped.

When using a DDS tape written with the current release of **fbackup** to do a partial recovery, **frecover** attempts to use the DDS fast-search capability to find files on the tape more quickly. In order to do this, however, **frecover** needs to create an in-memory copy of the index, and mark the files on that index which it needs to recover before actually reading through the tape to find the files. This is done when the first index is read from the tape, and accounts for a period of time just after recovery is begun where the tape is inactive while this in-memory index is constructed. The larger the index is, the longer this period lasts.

The utility set comprised of *fbackup* and **frecover** was originally designed for use on systems equipped with not more than one gigabyte of total file system storage. Although the utilities have no programming limitations that restrict users to this size, complete backups and recoveries of substantially larger systems can cause a large amount system activity due to the amount of virtual memory (swap space) used to store the indices. Users who want to use these utitilies, but are noticing poor system-wide performance due to the size of the backup, are encouraged to back up their systems in multiple smaller sessions, rather than attempting to back up the entire system at one time.

Note that when recovering files with access-control lists, the ACL entries are stored on the backup as user login names. If a login name cannot be found in the password file, the file is recovered without its ACL, and an error is printed. In order to fully recover files backed up with ACLs, the password file (**/etc/passwd**) must be recovered before attempting to recover any desired ACLs.

Care should be taken to match the names specified by the include and exclude options with the names in the index on the tape. Since the files are stored on the backup in lexographic order as defined by the **LANG** or **LC_COLLATE** environment variable, **frecover** uses the exact path names to determine when a partial recovery is complete, and when an earlier tape needs to be loaded. If a user's specification of a file to be recovered is misspelled, this may cause confusing messages, such as **frecover** asking for the previous volume, when volume one is mounted.

DEPENDENCIES

SS Series 700/800 **frecover** is not supported on QIC devices with QIC-120, and QIC-150 formats. If **frecover** is attempted for these formats, **frecover** fails and the following message is displayed :

```
mt lu X:Read must be a multiple of 512 bytes in QIC 120 and QIC 150
```

AUTHOR

frecover was developed by HP.

FILES

/dev/rmt/0m Default backup device.

SEE ALSO

cpio(1M), dump(1M), fbackup(1M), restore(1M), rmt(1M), tcio(1M), cdf(4), acl(5).

NAME

freeze - freeze sendmail configuration file on a cluster

SYNOPSIS

/etc/freeze [*sendmail options ...*]

DESCRIPTION

On a cluster, the **sendmail** frozen configuration file, **/usr/lib/sendmail.fc**, is a context-dependent file; **freeze** runs **/usr/lib/sendmail -bz** on each currently active cnode of the cluster in such a way that mail from any of the client cnodes appears as if it originated on the server cnode. This causes replies to all messages from the cluster to be directed to the server cnode, since only the server runs the **sendmail** daemon. On a stand-alone host, **freeze** simply execs **/usr/lib/sendmail -bz**.

freeze must be run by the super-user. Any arguments to **freeze** are appended to the **sendmail** command run on each cnode.

DIAGNOSTICS

freeze issues an error message if it is run by other than the super-user. Other error messages may be output by **remsh** or **sendmail**.

RETURN VALUE

If it is run by other than the super-user, **freeze** returns 1. Otherwise it returns the result of **/usr/lib/sendmail -bz** on the local host.

WARNINGS

freeze neither refreezes nor removes frozen configuration files belonging to inactive cnodes in the **/usr/lib/sendmail.fc+** CDF directory.

AUTHOR

freeze was developed by HP.

FILES

/etc/freeze *freeze* command object code

/usr/lib/sendmail.cf
default **sendmail** configuration file

/usr/lib/sendmail.fc
default frozen configuration file

SEE ALSO

cnodes(1), remsh(1), sendmail(1m), cdf(4).

NAME

fsck - file system consistency check and interactive repair

SYNOPSIS

/etc/fsck -p [**-F**] [*special ...*]

/etc/fsck -P [**-F**] [*special ...*]

/etc/fsck [**-b** *block#*] [**-y**] [**-n**] [**-F**] [**-q**] [*special ...*]

DESCRIPTION

fsck audits and interactively repairs inconsistent conditions for HP-UX file systems on mass storage device files identified by *special*. If the file system is consistent, the number of files on that file system and the number of used and free blocks are reported. If the file system is inconsistent, **fsck** provides a mechanism to fix these inconsistencies, depending on which form of the **fsck** command is used.

fsck checks a default set of file systems or the file systems specified in the command line. If *special* is not specified, **fsck** reads the table in **/etc/checklist**, using the first field (special-file name) to determine which file systems to check.

If the target device is a swap device, **fsck** does not continue to process. **fsck** also checks the target device to ensure a mounted file system is not being checked. If a mounted device is specified but the **-F** option is not, **fsck** prompts the user for a response.

If the **-p** option is used and *special* is not specified, **fsck** reads the specified pass numbers in **/etc/checklist** to inspect groups of disks in parallel, taking maximum advantage of I/O overlap to preen the file systems as quickly as possible. The **-p** option is normally used in the script **/etc/bcheckrc** during automatic reboot. Normally, the root file system is checked on pass 1, and other "root" ("0" section) file systems on pass 2. Other small file systems are checked on separate passes (such as the "section 4" file systems on pass 3 and the "section 7" file systems on pass 4), and finally the large user file systems are checked on the last pass (for example, pass 5). A pass number of zero or a type which is neither **rw** nor **ro** in **/etc/checklist** causes a file system not to be checked. If the optional fields are not present on a line in **/etc/checklist**, or the pass number is –1, **fsck** preens the file system on such lines sequentially after all eligible file systems with positive pass numbers have been preened.

Below are the inconsistencies that **fsck** with the **-p** option corrects. If it encounters other inconsistencies, it exits with an abnormal return status. For each corrected inconsistency, one or more lines are printed identifying the file system on which the correction will take place, and the nature of the correction. Correctible inconsistencies are limited to the following:

- Unreferenced inodes
- Unreferenced continuation inodes (see *inode*(4))
- Unreferenced pipes and fifos
- Link counts in inodes too large
- Missing blocks in the free list
- Blocks in the free list also in files
- Counts in the super-block wrong.

The **-P** option operates in the same manner as the **-p** option except that cleanly unmounted file systems are not checked (see *fsclean*(1M)). This can greatly decrease the amount of time required to reboot a system that was brought down cleanly.

Without the **-p** or **-P** option, **fsck** prompts for concurrence before each correction is attempted when the file system is inconsistent. It should be noted that some corrective actions result in a loss of data. The amount and severity of data loss can be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no.** If the operator does not have write permission, **fsck** defaults to a **-n** action.

Options

fsck recognizes the following options:

-b Use the block specified immediately after the flag as the super block for the file system. An alternate super block can always be found at block ((**SBSIZE+BBSIZE**)/**DEV_BSIZE**), typically block 16 (**DEV_BSIZE** is defined in <**sys/param.h**>).

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

`-F` Force `fsck` to check a mounted file system.

`-n` Assume a `no` response to all questions asked by `fsck`; do not open the file system for writing.

`-q` Quiet *fsck*. Do not print size-check messages in Phase 1. Unreferenced fifos are silently removed. If `fsck` requires it, counts in the superblock and cylinder groups are automatically fixed.

In all cases, `fsck` checks the following inconsistencies:

- Blocks claimed by more than one inode or the free list.
- Blocks claimed by an inode or the free list outside the range of the file system.
- Incorrect link counts.
- Size checks:
  - Directory size not of proper format.
- Bad inode format.
- Blocks not accounted for anywhere.
- Directory checks:
  - File pointing to unallocated inode.
  - Inode number out of range.
- Super Block checks:
  - More blocks for inodes than there are in the file system.
- Bad free block list format.
- Total free block and/or free inode count incorrect.
- Invalid continuation inode number in a primary inode.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the `lost+found` directory. The name assigned is the inode number. The only restriction is that the directory `lost+found` must already exist in the root of the file system being checked, and must have empty slots in which entries can be made. This is accomplished by making `lost+found`, copying a number of files to the directory, then removing them (before `fsck` is executed).

Unreferenced continuation inodes are removed with the `-p` option, since they do not refer back to the primary inode. When a primary inode contains an invalid continuation inode number, the continuation inode number should be cleared (that is, set to 0). This is not done automatically (with the `-p` option), because access control list information may have been lost and should be corrected.

After `fsck` has checked and fixed the file system, it stores the correct `fs_clean` flag in the super block if it is not already there. For a non-root file system, `FS_CLEAN` is stored there. For the root file system, which is mounted at the time of the `fsck`, no changes are required to the super block if no problems were found and `FS_OK` was already set.

Checking the raw device is almost always faster.

RETURN VALUE

`fsck` returns the following values:

`0` Either no errors were detected or all errors were corrected.

`4` Root file system errors were corrected. The system must be rebooted.

`8` Some uncorrected errors exist on one or more of the file systems checked, there was a syntax error, or some other operational error occurred.

12 A signal was caught during processing.

WARNINGS

`fsck` should not be run on mounted file systems or on the raw root device.

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

Be sure the system is in single-user state before running `fsck` (see *shutdown*(1M)).

DEPENDENCIES

Series 300/400/700

There is only one section per volume.

AUTHOR

`fsck` was developed by HP, AT&T, the University of California, Berkeley.

FILES

`/etc/checklist` contains default list of file systems to check.

SEE ALSO

dumpfs(1M), fsclean(1M), mkfs(1M), newfs(1M), shutdown(1M), checklist(4), fs(4), inode(4), acl(5).

NAME

fsclean - determine shutdown status of specified file system

SYNOPSIS

`/etc/fsclean [-q] [-v] [`*special* ... `]`

DESCRIPTION

`fsclean` determines the shutdown status of the the file system specified by *special* or, in the absence of *special*, the file systems listed in `/etc/checklist` of type `rw` or `ro`. All optional fields in `/etc/checklist` must be present for `fsclean` to be able to check each file system.

`fsclean` reads the super block to determine whether the file system's last shutdown was done correctly, and returns one of the following values:

- **0** All of the checked file systems were shut down correctly.
- **1** One or more checked file systems were not shut down correctly, implying that `fsck` should be run (see *fsck*(1M)).
- **2** Other error (such as `cannot open the specified device file`).

The `fsclean` command is usually silent.

Options:

`-q` (check quotas) Instead of checking the file system shutdown status, `fsclean` checks the validity of disk quota statistics. Only `/etc/checklist` entries of type `hfs` with the `rw` (or `default`) and `quota` options are checked. This option is useful for determining whether `quotacheck` should be run (see *quotacheck*(1M)).

`-v` Be verbose. Prints the status of each file system checked.

AUTHOR

`fsclean` was developed by HP.

FILES

`/etc/checklist`

SEE ALSO

brc(1M), dumpfs(1M), fsck(1M), mount(1M), quotacheck(1M), reboot(1M), checklist(4).

NAME

fsdb - file system debugger

SYNOPSIS

/etc/fsdb *special* [**-b** *block#*] [**-**]

Remarks

Always execute *fsck*(1M) after having running *fsdb*.

DESCRIPTION

fsdb can be used to patch up a damaged file system after a crash. It normally uses the first super block for the file system located at the beginning of the disk section as the effective super block. If the **-b** flag is used, the block specified immediately after the flag is used as the super block for the file system. An alternate super block will always be found at block ((SBSIZE + BBSIZE)/DEV_BSIZE), typically block 16.

fsdb deals with the file system in terms of block fragments, which are the unit of addressing in the file system and the minimum unit of space allocation. To avoid possible confusion, *fragment* is used to mean that, and *block* is reserved for the larger true block. *fsdb* has conversions to translate fragment numbers and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an inode. These greatly simplify the process of correcting control block entries or descending the file system tree.

fsdb contains several error-checking routines to verify inode and fragment addresses. These can be disabled if necessary by invoking *fsdb* with the optional - argument, or by the use of the **O** symbol.

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. Hexadecimal numbers must be prefixed with **0x**. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

fsdb reads a fragment at a time. A buffer management routine is used to retain commonly used fragments of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding fragment.

The following symbols are recognized by *fsdb*:

#	absolute address
i	convert from i-number to inode address (for continuation inodes as well as primary inodes; see *inode*(4))
b	convert from fragment number to disk address (historically "block")
d	directory slot offset
+,-	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=-	decremental assignment
="	character string assignment
X	hexadecimal flip flop
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double-word mode
!	escape to shell

The print facilities generate a formatted output in various styles. Octal numbers are prefixed with a zero. Hexadecimal numbers are prefixed with **0x**. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the interrupt character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect fragment boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current fragment are printed. The print options available are:

i	print as inodes (primary or continuation)

d	print as directories
o	print as octal words
x	print as hexadecimal words
e	print as decimal words
c	print as characters
b	print as octal bytes

The **f** symbol is used to print data fragments associated with the current inode. If followed by a number, that fragment of the file is printed. (Fragments are numbered from zero). The desired print option letter follows the fragment number, if present, or the **f** symbol. This print facility works for small as well as large files except for special files such as fifos, and device special files.

Dots, tabs, and spaces can be used as function delimiters but are not necessary. A line with just a new-line character increments the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry, or inode, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words, and double words are displayed with the octal (hexadecimal if X toggle is used) address followed by the value in octal (hexadecimal if X toggle is used) and decimal. A **.B** or **.D** is appended to the address for byte and double-word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. Inodes are printed with labeled fields describing each element.

The following mnemonics are used for inode examination and refer to the current working inode:

a#	data block numbers (0 - 14)
at	time last accessed
ci	continuation inode number
cno	cnode ID for a device file
ct	last time inode changed
gid	group ID number
ln	link count
md	mode
mt	time last modified
maj	major device number
min	minor device number
sz	file size in byte unit
uid	user ID number

The following mnemonics are used for directory examination:

di	i-number of the associated directory entry
nm	name of the associated directory entry

EXAMPLES

386i	prints i-number 386 in an inode format. This now becomes the current working inode.
ln=4	changes the link count for the working inode to 4.
ln=+1	increments the link count by 1.
fc	prints in ASCII fragment zero of the file associated with the working inode.
2i.fd	prints the first fragment-size piece of directory entries for the root inode of this file system.
d5i.fc	changes the current inode to that associated with the fifth directory entry (numbered from zero) found from the above command. The first fragment's worth of bytes of the file are then printed in ASCII.
1b.px	prints the first fragment of the superblock of this file system in hexadecimal.
2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
d7.nm="name"	changes the name field in the directory slot to the given string. Quotes are optional if the first character of the name field is alphabetic.
a2b.p0d	prints the third fragment of the current inode as directory entries.

WARNINGS

Use of *fsdb* should be limited to experienced *fsdb* users. Failure to understand fully the usage of *fsdb* and the file system's internal organization can lead to complete destruction of the file system and total loss of data.

AUTHOR

fsdb was developed by HP and AT&T.

SEE ALSO

dumpfs(1M), fsck(1M), stat(2), dir(4), fs(4).

NAME

fsirand - install random inode generation numbers

SYNOPSIS

`/etc/fsirand` [`-p`] *special*

DESCRIPTION

`fsirand` installs random inode generation numbers on all the inodes on device *special,* and also installs a filesystem ID in the superblock. This process increases the security of filesystems exported by NFS.

Use `fsirand` only on an unmounted filesystem that was checked with `fsck` (see *fsck*(1M)). The only exception is that it can be used on the root filesystem in single-user mode if the system is immediately re-booted afterwards using `reboot -n`.

The `-p` option prints the generation numbers for all inodes.

WARNINGS

`fsirand` should not be run on mounted filesystems. If executing `fsirand` on the root filesystem, the system should be in single-user mode and should be re-booted immediately afterwards using `reboot -n`.

AUTHOR

`fsirand` was developed by Sun Microsystems, Inc.

SEE ALSO

statfs(2).

INTERNATIONAL SUPPORT

messages.

(Requires Optional ARPA Services Software)

NAME

ftpd - DARPA Internet File Transfer Protocol server

SYNOPSIS

/etc/ftpd [**-l**] [**-t** *timeout*] [**-T** *maxtimeout*] [**-u** *umask*]

DESCRIPTION

ftpd is the DARPA Internet File Transfer Protocol server. It expects to be run by the Internet daemon; see *inetd*(1M) and *inetd.conf*(4). *inetd* runs *ftpd* when a service request is received at the port indicated in the **ftp** service specification in **/etc/services**; see *services*(4).

The **-l** option causes each FTP session to be logged in the syslog.

The **-t** option causes *ftpd* to timeout inactive sessions after *timeout* seconds. By default, *ftpd* will timeout an inactive session after 15 minutes.

A client can also request a different timeout period. The **-T** option sets to *timeout* the maximum timeout that client can request. By default, the maximum timeout is 2 hours.

By default, *ftpd* sets its umask to 027. To change this default umask, use the **-u** option.

ftpd currently supports the following commands (uppercase and lowercase are interpreted as equivalent):

Command	**Description**
ABOR	Abort previous command
ACCT	Specify account (ignored)
ALLO	Allocate storage (vacuously)
APPE	Append to a file
CDUP	Change to parent of current working directory
CWD	Change working directory
DELE	Delete a file
HELP	Give help information
LIST	Give list files in a directory (**ls -l**)
MKD	Make a directory
MDTM	Show last modification time of file
MODE	Specify data transfer *mode*
NLST	Give name list of files in directory
NOOP	Do nothing
PASS	Specify password
PASV	Prepare for server-to-server transfer
PORT	Specify data connection port
PWD	Print the current working directory
QUIT	Terminate session
REST	Restart incomplete transfer
RETR	Retrieve a file
RMD	Remove a directory
RNFR	Specify rename-from file name
RNTO	Specify rename-to file name
SITE	Non-standard commands (see next section)
SIZE	Return size of file
STAT	Return status of server
STOR	Store a file
STOU	Store a file with a unique name
STRU	Specify data transfer *structure*
SYST	Show operating system type of server system
TYPE	Specify data transfer *type*
USER	Specify user name
XCUP	Change to parent of current working directory
XCWD	Change working directory
XMKD	Make a directory
XPWD	Print the current working directory
XRMD	Remove a directory

(Requires Optional ARPA Services Software)

The following non-standard or HP-UX specific commands are supported by the SITE command:

Command	Description
UMASK	Change umask. (e.g., SITE UMASK 002)
IDLE	Set idle-timer. (e.g., SITE IDLE 60)
CHMOD	Change mode of a file. (e.g., SITE CHMOD 755 filename)
HELP	Give help information. (e.g., SITE HELP)

The remaining ftp requests specified in Internet RFC 959 are recognized, but not implemented. MDTM and SIZE are not specified in RFC 959, but are expected in the next updated FTP RFC.

The FTP server aborts an active file transfer only when the ABOR command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in Internet RFC 959. If *ftpd* receives a STAT command during a data transfer, preceded by a Telnet IP and Synch, it returns the status of the transfer.

ftpd interprets file names according to the "globbing" conventions used by *csh*(1). This allows users to utilize the metacharacters *, ., [,], {, }, ~, and **?**.

ftpd authenticates users according to three rules:

- The user name must be in the password data base, **/etc/passwd**, and not have a null password. The client must provide the correct password for the user before any file operations can be performed.
- The user name must not appear in the file **/etc/ftpusers** (see *ftpusers*(4)).
- The user must have a standard shell returned by *getusershell*(3).

Optionally, a system administrator can permit public access or "anonymous FTP." If this has been set up, users can access the anonymous FTP account with the user name **anonymous** or **ftp** and any non-null password (by convention, the client host's name). *ftpd* does a *chroot*(2) to the home directory of the user **ftp**, thus limiting anonymous FTP users' access to the system. If the user name is **anonymous** or **ftp**, an anonymous FTP account must be present in the password file (user **ftp**). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In order to permit anonymous FTP, there must be an entry in the *passwd*(4) database for an account named **ftp**. The password field should be *, the group membership should be **guest**, and the login shell should be **/bin/false**. For example (assuming the **guest** group ID is **10**):

ftp:*:500:10:anonymous ftp:/users/ftp:/bin/false

The anonymous ftp directory should be set up as follows:

~ftp The home directory of the ftp account should be owned by user **ftp** and mode 555 (not writable). Since *ftpd* does a *chroot* to this directory, it must have the following subdirectories and files:

~ftp/bin This directory must be owned by root and mode 555 (not writable). It should contain a copy of **/bin/ls**. This is needed to support directory listing by *ftpd*. The command should be mode 111 (executable only). If the FTP account is on the same file system as **/bin**, **~ftp/bin/ls** can be hard link, but it may not be a symbolic link, because of the *chroot*. The command must be replaced when the system is updated.

~ftp/etc This directory must be owned by root and mode 555 (not writable). It should contain versions of the files *passwd*, *group*, and *logingroup*. See *passwd*(4) and *group*(4). These files must be owned by root and mode 444 (readable only). These are needed to map user and group ids in the LIST command, and to support (optional) sub-logins of anonymous FTP. Sub-logins can sometimes be used to allow access to particular files by only specific remote users (who know the sub-login password) without giving those remote users logins on the system. A sub-login user would access the system via anonymous ftp, and then use USER and PASS to change to the sub-login user.

~ftp/etc/passwd
This file should contain entries for the **ftp** user and any other users who own files under the anonymous **ftp** directory. Such entries should have * for passwords. **~ftp/etc/passwd** should also contain entries for any desired anonymous FTP sub-logins. The sub-logins must have passwords, which must be encrypted as in *passwd*(4).

Group IDs must be listed in the anonymous FTP group file, **~ftp/etc/group**. The path names of home directories in **~ftp/etc/passwd** must be with respect to the anonymous FTP home directory. A sub-login home directory should be owned by the sub-login user ID. The shell field is ignored, and can be empty.

For example, the anonymous FTP sub-login name **subftp** would have an entry in the FTP **passwd** file that resembles:

subftp:bAg6vI82aq5Yt:501:10:ftpsub-login:/subftp:

FTP sub-login IDs do not need to be present in the system **/etc/passwd** file. Assuming the anonymous FTP directory is **/users/ftp**, the sub-login home directory in the example would be created by user **root** as follows:

```
cd /users/ftp
mkdir subftp
chmod 700 subftp
chown 501 subftp
chgrp guest subftp
```

File **~ftp/etc/group** should contain the group names associated with any group IDs in file **~ftp/etc/passwd** and any group IDs of files in the anonymous FTP subdirectories. In the above example, **~ftp/etc/group** would require an entry for **guest**, and the associated group ID would have to be the same as in the system's **/etc/group** file.

~ftp/etc/logingroup
Permits anonymous ftp sub-logins to be members of multiple groups. Can be a hard link to FTP **~ftp/etc/group**.

~ftp/pub (optional)
This directory is used by anonymous FTP users to deposit files on the system. It should be owned by user **ftp** and should be mode 777 (readable and writable by all).

~ftp/dist (optional)
Directories used to make files available to anonymous ftp users should be mode 555 (not writable), and any files to be distributed should be owned by root and mode 444 (readable only) so that they cannot be modified or removed by anonymous FTP users.

DIAGNOSTICS

ftpd replies to FTP commands to ensure synchronization of requests and actions during file transfers, and to indicate the status of *ftpd*. Every command produces at least one reply, although there may be more than one. A reply consists of a three-digit number, a space, some text, and an end of line. The number is useful for programs; the text is useful for users. The number must conform to this standard, but the text can vary.

The first digit of the message indicates whether the reply is good, bad, or incomplete. Five values exist for the first digit. The values and the interpretations of the values are:

1 The requested action is being initiated; expect another reply before proceeding with a new command.

2 The requested action is complete. The server is ready for a new request.

3 The command has been accepted, but the requested action requires more information.

4 The command was not accepted, the requested action failed, but the error condition is temporary and the action can be requested again.

5 The command was not accepted, the requested action failed, and the error condition would most likely occur again if the same command sequence is repeated.

The second digit indicates the functional area that the message addresses. The values of the second digit and the interpretations of these values are:

0 Syntax. A message with a 0 for the second digit indicates that a syntax error occurred.

1 Information. A message with a 1 as the second digit indicates that the message is in reply to a request for information.

2 Connections. A message with a 2 as the second digit indicates that the message is a reply to a request for control and data connection information.

3 Authentication and accounting. A message with a 3 as the second digit indicates that the message is a reply to a login or accounting procedure.

4 Not currently specified.

5 File system. A message with a 5 as the second digit indicates that the text following the number contains information concerning the status of the server file system.

The third digit provides a further clarification of the information supplied by the second digit. Following are several examples of messages. Note that *ftpd*'s replies match the number but not the text.

110 Restart marker reply. MARK *yyyy* = *mmmm* where *yyyy* is a user process data stream marker, and *mmmm* is *ftpd*'s equivalent marker
119 Terminal not available, will try mailbox
120 Service ready in *nnn* minutes
200 Command okay
211 System status, or system help reply
212 Directory status
230 User logged in, proceed
250 Requested file action okay, completed
331 User name okay, need password
350 Requested file action pending further information
425 Cannot open data connection
451 Requested action aborted: local error in processing
500 Syntax error, command unrecognized or command line too long
530 Not logged in
550 Requested action not taken; file unavailable, not found, no access

WARNINGS

The password is sent unencrypted through the socket connection.

Anonymous FTP is inherently dangerous to system security.

An error in the treatment of carriage returns in FTP clients and servers based on the 4.2BSD implementation (specifically, any *ftp*(1) or *ftpd*(1M) released prior to HP-UX 8.0) has been corrected. This correction may result in incorrect transfers of binary files when using the **ascii** transfer type. Avoid this problem by using the **image** (**binary**) transfer type.

AUTHOR

ftpd was developed by the University of California, Berkeley.

SEE ALSO

ftp(1), inetd(1M), chroot(2), getusershell(3), inetd.conf(4), ftpusers(4), passwd(4), group(4).

NAME

fuser, cfuser - list process IDs of all processes that have *file* open

SYNOPSIS

/etc/fuser [**-ku**] *files* [**-**] [[**-ku**] *files*]

/etc/cfuser [**-ku**] *files* [**-**] [[**-ku**] *files*]

DESCRIPTION

fuser lists the process IDs of processes that have *file* open. For block special devices, all processes using any file on that device are listed. The process ID is followed by **c** if the process is using the file as its current directory or **p** if using the file as its root directory.

If the **-u** option is specified, the login name (in parentheses) also follows the process ID. In addition, if the **-k** option is specified, the **SIGKILL** signal is sent to each process. Only users with appropriate privileges can terminate another user's process (see *kill*(2)). Options can be respecified between groups of files. The new set of options replaces the old set, with a lone dash canceling any options currently in force.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

In the HP Clustered environment, **cfuser** identifies processes running on any node in the HP Cluster using a file or file structure. A separate report is produced for each member of the cluster. Each report is preceded by the cluster member's name.

EXAMPLES

fuser -ku /dev/dsk/1s?
if typed by a user with appropriate privileges, terminates all processes that are preventing disk drive 1 from being unmounted, listing the process ID and login name of each process being killed.

fuser -u /etc/passwd
lists process IDs and login names of processes that have the password file open.

fuser -ku /dev/dsk/1s? -u /etc/passwd
if typed by a user with appropriate privileges, combines both of the above examples into a single command line.

FILES

/dev/kmem	for system image
/dev/mem	also for system image
/hp-ux	for name list

SEE ALSO

mount(1M), ps(1), kill(2), signal(2).

STANDARDS CONFORMANCE

fuser: SVID2

NAME

fwtmp, wtmpfix - manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [**-ic**]
/usr/lib/acct/wtmpfix [*files*]

DESCRIPTION

fwtmp

fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in **wtmp** to formatted ASCII records. The ASCII version is useful to enable editing, via *ed*(1), bad records or general purpose maintenance of the file.

The argument **-ic** is used to denote that input is in ASCII form, and output is to be written in binary form. (The arguments **i** and **c** are independent, respectively specifying ASCII input and binary output, thus **-i** is an ASCII to ASCII copy and **-c** is a binary to binary copy).

wtmpfix

wtmpfix examines the standard input or named files in **wtmp** format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A - can be used in place of *files* to indicate the standard input. If time/date corrections are not performed, *acctcon1* will fault when it encounters certain date-change records.

Each time the date is set, a pair of date change records is written to **/etc/wtmp**. The first record is the old date denoted by the string **old time** placed in the line field and the flag **OLD_TIME** placed in the type field of the **<utmp.h>** structure. The second record specifies the new date, and is denoted by the string **new time** placed in the line field and the flag **NEW_TIME** placed in the type field. *wtmpfix* uses these records to synchronize all time stamps in the file. *wtmpfix* nullifies date change records when writing to the standard output by setting the time field of the **<utmp.h>** structure in the old date change record equal to the time field in the new date change record. This prevents *wtmpfix* and *acctcon1* from factoring in a date change record pair more than once.

In addition to correcting time/date stamps, *wtmpfix* checks the validity of the name field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it changes the login name to **INVALID** and write a diagnostic to the standard error. This minimizes the risk that *acctcon1* will fail when processing connect accounting records.

DIAGNOSTICS

wtmpfix generates the following diagnostics messages:

Cannot make temporary: xxx failed to make temp file
Input truncated at offset: xxx missing half of date pair
New date expected at offset: xxx missing half of date pair
Cannot read from temp: xxx some error reading
Bad file at offset: xxx ut_line entry not digit, alpha, nor | or { (first character only checked)
Out of core: *malloc* fails. (Saves table of date changes)
No dtab: software error (rarely seen, if ever)

FILES

/usr/include/utmp.h
/etc/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctcom(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), ed(1), runacct(1M), acct(2), acct(4), utmp(4).

BUGS

fwtmp generates no errors, even on garbage input.

STANDARDS CONFORMANCE

fwtmp: SVID2

wtmpfix: SVID2

NAME

gated - gateway routing daemon

SYNOPSIS

gated [**-c**] [**-n**] [**-t***trace_options*] [**-f** *config_file*] [*trace_file*]

DESCRIPTION

gated is a routing daemon that handles the RIP, BGP, EGP, and HELLO routing protocols. The **gated** process can be configured to perform all routing protocols or any combination of the four (see WARNINGS below).

Options

gated recognizes the following options and command-line arguments:

-c Parse the configuration file for syntax errors then exit. If there were no errors, leave a dump file in **/usr/tmp/gated_dump**. Running **gated** does not require super-user privilege when using the **-c** option, but **gated** may not be able to read the kernel's routing table unless it is run as super user. The **-c** option implies **-tierk**.

-n Do not modify the kernel's routing table. This option is used for testing **gated** configurations with actual routing data.

-t*trace_options*

Enable trace flags on startup. If no flags are specified, **ier** is assumed. No spaces are allowed between this option and its arguments.

This option must be used to trace events that occur before the configuration file is parsed, such as determining interface configuration and reading routes from the kernel.

trace_options can include one or more of the following values:

A	all	**P**	protocol
i	internal	**u**	update
e	external	**R**	RIP
k	kernel	**H**	hello
r	route	**C**	icmp
m	mark	**p**	EGP
t	nostamp	**B**	BGP

Trace options are explained in greater detail in the *gated-config*(4) manual entry.

-f *config_file*

Use an alternate configuration file. By default, **gated** uses **/etc/gated.conf**.

trace_file

Trace file in which to place trace information.

If no **-t** option is specified, or if a *trace_file* argument is specified, **gated** detaches from the terminal and runs in the background. If a **-t** option and accompanying trace flags are specified without specifying a trace file, **gated** assumes that tracing is to be sent to the console, and remains in the foreground.

Signal Processing

gated catches the following signals and processes them as indicated:

SIGHUP Re-read configuration.

A **SIGHUP** causes **gated** to reread the configuration file. **gated** first performs a clean-up of all allocated policy structures. All BGP and EGP peers are flagged for deletion and the configuration file is re-parsed.

If the re-parse is successful, any BGP and EGP peers that are no longer in the configuration are shut down, and new peers are started. **gated** attempts to determine whether changes to existing peers require a shutdown and restart.

It should also be possible to enable or disable any protocol without restarting **gated**.

SIGINT Snapshot of current state.

The current state of all **gated** tasks, timers, protocols, and tables are written to **/usr/tmp/gated_dump**.

On systems supporting **fork()**, this is done by forking a subprocess to dump the table information so as not to impact **gated**'s routing functions. On systems where memory management does not support copy-on-write, this causes the **gated** address space to be duplicated which may be cause a noticable impact on the system. On systems not supporting **fork()**, the main process immediately processes the dump, which may impact **gated**'s routing functions.

SIGTERM Graceful shutdown.

On receipt of a **SIGTERM**, **gated** attempts a graceful shutdown. All tasks and protocols are asked to shut down. Most terminate immediately, the exception being EGP peers which wait for confirmation. It may be necessary to repeat the **SIGTERM** once or twice if it this process takes too long.

All exerior routes (BGP and EGP) are removed from the kernel's routing table on receipt of a **SIGTERM**. Interior routes (all others) remain. To terminate **gated** with the exterior routes intact, use **SIGKILL** or **SIGQUIT** (which causes a core dump).

SIGUSR1 Toggle tracing.

On receipt of a **SIGUSR1**, **gated** closes the trace file. A subsequent **SIGUSR1** causes it to be reopened. This allows the file to be moved regularly.

It is not possible to use **SIGUSR1** if a trace file has not been specified, or if trace output is being sent to the console.

WARNINGS

gated contains provisions for BGP protocol, but it is not officially supported by HP at the present time.

AUTHORS

Mark Fedor, PSI
Jeffrey C Honig, Cornell University

SEE ALSO

arp(1m), ifconfig(1m), netstat(1m), gated-config(4).

RFC 891 DCN Local-Network Protocols (HELLO)
RFC 904 Exterior Gateway Protocol Formal Specification
RFC 1058 Routing Information Protocol
RFC 1163 A Border Gateway Protocol (BGP)
RFC 1164 Application of the Border Gateway Protocol in the Internet

NAME

getty - set terminal type, modes, speed, and line discipline

SYNOPSIS

/etc/getty [**-h**] [-t *timeout*] *line* [*speed* [*type* [*linedesc*]]]
/etc/getty -c *file*

DESCRIPTION

getty is a program that is invoked by *init*(1M). It is the second process in the series, (*init-getty-login-shell*) that ultimately connects a user with the HP-UX system. Initially, if **/etc/issue** exists, *getty* prints its contents to the user's terminal, followed by the login message field for the entry it is using from **/etc/gettydefs**. *getty* reads the user's login name and invokes the *login*(1) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used.

Configuration Options and Arguments

getty recognizes the following arguments:

line Name of a tty line in **/dev** to which *getty* is to attach itself. *getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing. By default *getty* forces a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. However, when *getty* is run on a direct port, *getty* does not force a hangup on the line since the driver ignores changes to zero speed on ports open in direct mode (see *modem*(7)).

-h Tells *getty* not to force a hangup on the line before setting the speed to the default or specified speed.

-t *timeout* *getty* exits if the open on the line succeeds and no one types anything within *timeout* seconds.

speed A label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* at what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by typing a *break* character). The default *speed* is 300 baud.

type A character string describing to *getty* what type of terminal is connected to the line in question. *getty* understands the following types:

none	default
vt61	DEC vt61
vt100	DEC vt100
hp45	Hewlett-Packard HP2645
c100	Concept 100

The default terminal is **none**; i.e., any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition.

linedesc

A character string describing which line discipline to use when communicating with the terminal. Hooks for line disciplines are available in the operating system, but there is only one presently available — the default line discipline, **LDISC0**.

When given no optional arguments, *getty* sets the *speed* of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, new-line characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This causes *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(2)).

The user's name is scanned to see if it contains any lowercase alphabetic characters; if not, and if the name is non-empty, the system is told to map any future uppercase characters into the corresponding lowercase characters.

getty also understands the "standard" ESS2 protocols for erasing, killing and aborting a line, and terminating a line. If *getty* sees the ESS erase character, _, or kill character, **$**, or abort character, **&**, or the ESS line terminators, / or !, it arranges for this set of characters to be used for these functions.

Finally, *login* is called with the user's name as an argument. Additional arguments can be typed after the login name. These are passed to *login*, which places them in the environment (see *login*(1)).

Check Option

A check option is provided. When *getty* is invoked with the **-c** option and *file*, it scans *file* as if scanning **/etc/gettydefs** and prints the results on the standard output. If there are any unrecognized modes or improperly constructed entries, *getty* reports these. If the entries are correct, *getty* prints out the values of the various flags. See *ioctl*(2) for an interpretation of values. Note that some values are added to the flags automatically.

DEPENDENCIES

HP2334 MultiMux:

The modem control parameter *MRTS* must be present in the **/etc/gettydefs** file when using *getty* in conjunction with an HP2334 or HP2335 MultiMux to ensure that the RTS modem control signal is asserted correctly.

Example:

```
9600# B9600 HUPCL PARENB MRTS # B9600 SANE PARENB ISTRIP IXANY #login: #19200
```

MRTS is not intended for use with devices other than the HP2334 or HP2335 MultiMux.

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

ct(1), login(1), init(1M), ioctl(2), gettydefs(4), inittab(4), modem(7), termio(7).

BUGS

While *getty* does understand simple single character quoting conventions, it is not possible to quote the special control characters that *getty* uses to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. Therefore it is not possible to log in by means of *getty* and type a **#**, **@**, **/**, **!**, **_**, backspace, **^U**, **^D**, or **&** as part of your login name or arguments. They will always be interpreted as having their special meaning as described above.

NAME

getx25 - get x25 line

SYNOPSIS

/etc/getx25 *line speed pad-type*

DESCRIPTION

getx25 is functionally very similar to **getty** (see *getty*(1M)) but is used only for incoming lines that are connected to an X.25 PAD. It performs special functions such as setting up an initial PAD configuration. It also logs the number of the caller in **/usr/spool/uucp/.Log/X25LOG**. The third parameter is the name of the PAD being used. HP2334A is the only one supported at this time. A typical invocation would be:

```
/etc/getx25 x25.1 2 HP2334A
```

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

DEPENDENCIES

Series 300/400

The log file used is **/usr/spool/uucp/X25LOG**.

AUTHOR

getx25 was developed by HP.

SEE ALSO

getty(1M), login(1), uucp(1).

UUCP tutorial in *Remote Access Users Guide*.

NAME

glbd - Global Location Broker Daemon

SYNOPSIS

/etc/ncs/glbd [**-create** { **-first** [**-family** *family_name*] | **-from** *host_name* }] [**-change_family** *family_name*] [**-listen** *family_list*] [**-version**]

DESCRIPTION

The Global Location Broker (GLB), part of the Network Computing System (NCS), helps clients locate servers on a network or internet. The GLB database stores the locations (that is, the network addresses and port numbers) where server processes are running. A daemon maintains this database and provides access to it.

There are two versions of the GLB daemon: *glbd* and *nrglbd*. The replicatable version, *glbd*, is provided only for Domain/OS, HP-UX, SunOS, and ULTRIX systems. For other systems, the non-replicatable version, **nrglbd**, is provided. The two versions of the daemon should not coexist on a given network. (For HP-UX systems, which may have both *glbd* and **nrglbd**, the use of only *glbd* is strongly recommended; only *glbd* is described in this manual entry.

The GLB database can be replicated to increase its availability. Copies of the database can exist on several hosts, with a *glbd* running on each of those hosts to maintain the consistency of the database replicas. (In an internet, at least one *glbd* must run in each network.) Each replica of the GLB keeps a list of all the other GLB replicas. *drm_admin*(1M) administers the replication of the GLB database and the replica list.

Currently, *glbd* supports both the DARPA IP and Domain DDS network protocols. A GLB replica can allow access to its database from both IP and DDS clients. However, when communicating with each other to maintain replication of the GLB database, GLB replicas should use only one protocol family. You choose which family the GLBs will use. In an internet, all routing nodes must support this family.

If a set of GLB replicas includes any HP-UX, SunOS, or ULTRIX systems, all replicas must use IP protocols to communicate with each other. A replica running on a Domain/OS system can communicate with other replicas via IP protocols but still provide lookup and update services to its clients via both IP and DDS protocols.

Running glbd on HP-UX Systems

On HP-UX systems, the GLB communicates only via IP protocols. A Local Location Broker daemon (*llbd*) must be running on the local host when *glbd* is started.

Typically, the *llbd* and *glbd* processes are started in background at boot time from **/etc/netncsrc**; if the **START_GLBD** variable in **netncsrc** is set to 1, a *glbd* will be started. To start the *llbd* and *glbd* daemons by hand, you must be root.

OPTIONS

-create Create a replica of the GLB. This option creates a GLB database in addition to starting a broker process. It must be used with either **-first** or **-from**.

-first This option can be used only with the **-create** option. Use it to create the first replica (that is, the very first instance) of the GLB on your network or internet.

-family *family_name*

This option can be used only in conjunction with the **-first** option. It specifies the address family that the first GLB replica will use to identify itself on the replica list. Any subsequently created replicas must use this family to communicate with this replica. Currently, *family_name* can be either **ip** or **dds**. If this option is not used, the replica will be identified on the replica list by its IP address.

-from *host_name*

This option can be used only with the **-create** option. Use it to create additional replicas of the GLB. A replica of the GLB must exist at *host_name*. The database and replica list for the new replica are initialized from those at *host_name*. The replica at *host_name* adds an entry for the new replica to its replica list and propagates the entry to the other GLB replicas.

A *host_name* takes the form *family*:*host*, where the host can be specified either by its name or by its network address. For example, **ip:bertie**, **ip:#192.5.5.5**, **dds://jeeves**, and **dds:#959a.940f** are acceptable host names.

The new replica will use the same address family as *host_name* in identifying itself on the replica list. For example, if *host_name* is an IP address, the new replica will be listed by its IP address on the replica list.

-change_family *family_name*
Use this option only if network reconfigurations require that you change the address family of every GLB replica; see the discussion in the DESCRIPTION section. Currently, *family_name* can be either **ip** or **dds**. (If a GLB replica runs on an HP-UX host, its address family must be IP.)

For a procedure to change all of your GLB replicas from one address family to another, see *Managing NCS Software*.

-listen *family_list*
This option restricts the address families on which a GLB listens. Use it only if you are creating a special configuration where access to a GLB is restricted to a subset of hosts in the network or internet.

The *family_list* is a list of the address families on which the GLB will listen. Names in this list are separated by spaces. Possible family names include **ip** and **dds**.

The GLB will always listen for requests from the family by which it is listed on the replica list, even if that family is not specified in *family_list*.

If *glbd* is started without the **-listen** option, the GLB will listen on all address families that are supported both by NCS and by the local host. On Domain/OS systems, this set of families always includes **dds** and may also include **ip**. On most other systems, including HP-UX systems, **ip** is currently the only family.

-version Display the version of NCK that this *glbd* belongs to, but do not start the daemon.

EXAMPLES

Create and start for the first time the first replica of the GLB on this network or internet:

```
$ /etc/ncs/glbd -create -first &
```

Start for the first time a subsequent replica of the GLB, initializing its database from host **jeeves**:

```
$ /etc/ncs/glbd -create -from ip:jeeves &
```

Restart an existing replica of the GLB:

```
$ /etc/ncs/glbd &
```

FILES

On Domain/OS systems, *glbd* writes diagnostic output to the file `**node_data/system_logs/glb_log**. On other UNIX systems, including HP-UX systems, the log file is **/etc/ncs/glb_log**.

SEE ALSO

drm_admin(1M), lb_admin(1M), llbd(1M), glb_obj.txt(4), glb_site.txt(4).

Managing NCS Software.

(Requires Optional ARPA Services Software)

NAME

hosts_to_named - Translate host table to name server file format

SYNOPSIS

hosts_to_named -d *domain* **-n** *network-number* [*options*]

DESCRIPTION

hosts_to_named translates the host table, */etc/hosts*, into files that are usable by the name server *named*(1M). The format of these files is defined in RFC1035. The files are created in the current directory. Once the host table is translated, the name server files can be maintained directly, or the translation can be repeated after each change to the host table.

If a line in the host table contains no domain names, all names on the line are assumed to be in the default domain. The first *domain* listed is the "default domain". If data is being created for more than 1 domain or if certain options are used, there must be domain names in the host table to determine which names belong in which domain.

The name server data is referred to as "resource records".

Options are:

-a *network-number*
Add the information about hosts in the local domain from network *network-number*. This is the same as the **-n** option except that no pointer (PTR) data is created. This is useful when there are multiple domains on a network and a different server is handling the address-to-name mapping for *network-number*.

-b *bootfile* Name the boot file *bootfile*. The default is **named.boot** in the current directory.

-c *subdomain* Create alias (CNAME) records for hosts in *subdomain* of the default domain. When a subdomain is delegated, it is useful to create aliases for the old names in the default domain that point to the new names in the *subdomain*. After creating the alias (CNAME) records, ignore lines in the host table that contain names in the *subdomain*. This option can be used more than once on the command line. This option requires domain names in the host table. When the old names in this *domain* are no longer used, they can be ignored with the **-e** option. If the *subdomain* name does not have dots, the default domain is appended to *subdomain*.

-d *domain* Create data for *domain*. This option can be used more than once on the command line if data is being created for more than 1 domain. The first *domain* listed is the "default domain". This option requires domain names in the host table for all hosts in domains except the default domain.

-e *subdomain* Eliminate lines from the host table that contain names in the *subdomain* before translating. If the *subdomain* name does not have dots, the default domain is appended. This option may be used more than once on the command line. This option requires domain names in the host table.

-f *file* Read command line options from *file*. The **-f** option is not allowed within a file.

-h *host* Declare *host* to be the host in the start of authority (SOA) record that the name server data was created on. Also use *host* for the electronic mail address of the responsible user in the SOA record. The default is the host this command is run on.

-m *weight:mailhub*
For each canonical hostname from the host table, create mail exchanger (MX) records with the specifed weight and mail hub. The weight is a positive integer. The mail hub is a hostname. If the mail hub name has no dots, the default domain is appended. This option can be used more than once on the command line.

-n *network-number[:mask]*
Create data for *network-number*. See below for description of *network-number*. If only one *domain* is listed with **-d**, all data for *network-number* is assumed to be in *domain*. The optional subnet mask *mask* can be used instead of supplying each *network-number* for a subnet using multiple **-n** options. *mask* must be in dot notation.

(Requires Optional ARPA Services Software)

-o *refresh:retry:expire:min*
Set the values in the start-of-authority (SOA) record to those specified. See below for description of the start-of-authority (SOA) record.

-p *domain* Create only pointer (PTR) data for hosts in *domain*. This is useful when there are multiple domains on a network and a different server is responsible for *domain*, but this server is responsible for the address-to-name mapping. This option can be used more than once on the command line. This option requires domain names in the host table.

-q Run quietly. No messages are printed.

-r Create name server data indicating that the name server is authoritative for . (the root of the domain tree). The file created is **db.root**. Use this only when your network is isolated from the ARPA Internet. If other root servers exist for the isolated network, they must be added manually.

-s *server* Create name server (NS) records that declare *server* is an authoritative name server for all of the domains created. If more than 1 server is authoritative, each needs to be declared. If the server name does not have any dots in it, the default domain is appended. The default server is the host this script is run on. This option can be used more than once on the command line.

-t Create text (TXT) records from the comments that appear with host data. The comments will all be in lower case because the host table is translated to lower case. If **[no smtp]** appears in a comment, it is omitted. The **[no smtp]** is used to control mail exchanger (MX) data.

-u *user* Declare *user* to be the electronic mail address of the person responsible for this domain.
This is used in the start of authority (SOA) record. The format required in the name server data is *user*.**host** (host must be a domain name). If given as *user*, the host on which this script is run is appended. If given as *user@host*, the @ is replaced with a dot (.). The default user is **root**.

-w Create well known services (WKS) data declaring that the host provides the SMTP service. This is done only when mail exchanger (MX) data is also being created and only for hosts without **[no smtp]** in a comment.

-z *internet-address*
Create a secondary boot file, **boot.sec.save**, from the primary boot file listing *internet-address* as the server to load the data from. The boot file has the server back up the data on disk. The *internet-address* defaults to the value used with **-Z**. This option can be used more than once.

-A Do not create name server data for aliases in the host table.

-C *file* Create resource records from strings in the comment field of the host table. Each string in the comment field (except **[no**smtp]) is searched for in *file*. The format of *file* is a string, a colon, and a resource record. If the string in the comment field matches the string before the colon in *file*, a resource record is added consisting of the name of the host followed by everything after the colon from the matching line in *file*. For example, host information (HINFO) records can be created by adding **360:IN HINFO hp9000s360 hp-ux** to *file* and adding 360 to comments in the host table.

-D Do not create name server data for domain names in the host table.

-F By default, the serial number is incremented for a domain only if the data has changed (pointer (PTR) data only). This option forces the serial number to be incremented, even if the data has not changed.

-H *host-file* Use *host-file* instead of **/etc/hosts**.

-M Do not create mail exchanger (MX) records for hosts in the host table.

-N *mask* Apply the default subnet mask *mask* to each *network-number* specified with **-n** except for ones with their subnet masks already provided. *mask* must be in dot notation.

This is the same as supplying each *network-number* for a subnet using multiple **-n** options.

-S *server* This option is the same as the **-s** option, but it only applies to the last *domain* specified with **-d** or the last *network-number* specified with **-n**. This option is for when *server* is backing up some, but not all, of the domains.

-Z *internet-address*
Create a secondary boot file, **boot.sec**, from the primary boot file listing *internet-address* as the server to load the data from. The boot file does not have the server back up the data on disk. The *internet-address* defaults to value used with **-z**. This option can be used more than once.

-1 This option is obsolete.

hosts_to_named translates the host table to lower case to help eliminate duplicate data. Since the name server treats uppercase and lowercase as equivalent, names that differ only in case are considered the same.

Alias (CNAME) records are created for *subdomains* delegated with **-c**. Lines from the host table that contain names in *subdomains* from **-c** and **-e** are removed from the lowercase copy of the host table.

The host table is then used to create the name server data for each *network-number* declared on the command line. Do not include the trailing 0's in the network number. No distinction is made between class A, B, or C addresses nor is there any understanding of subnets unless a subnet mask is supplied. Example network numbers are: 10 (for all addresses of the form 10.*.*.*), 10.1 (for addresses of the form 10.1.*.*), or 10.2.2 (for addresses of the form 10.2.2.*).

Address (A) records are created for mapping hostnames to IP addresses. Alias (CNAME) records are created for aliases of hosts that are not multi-homed. The data are placed in a file named **db.***DOMAIN* where *DOMAIN* is the first part of the domain from the command line. For the domain **div.inc.com**, the file is named **db.div**. All other name server data goes in this file except the pointer (PTR) records described below.

Pointer (PTR) records are created for mapping IP addresses to host names. PTR records are placed in a file named **db.***NET* where *NET* is the network number from the command line. Network 10 data is placed in **db.10**. Network 10.1 data are placed in "db.10.1".

Mail exchanger (MX) records are created unless the **-M** option is used. The default MX record has a weight of 10 with the host itself as its mail exchanger. No default MX record is created for a host if **[no smtp]** is in the comment section of that line in the host table. MX records for each mail hub declared with the **-m** option are added for each host even if **[no smtp]** is in the comment section.

Well known services (WKS) records are created for each host that handles SMTP mail (does not have **[no**smtp]) if **-w** is used. The only service listed is SMTP.

Text (TXT) records are created for comments associated with hosts in the host table if **-t** is used. The comments do not include **[no**smtp].

For each domain, a start of authority (SOA) record is created. The SOA record requires 2 domain names: the host that the data is created on and the electronic mail address of the person responsible. The **-h** and **-u** options influence the names. In addition, the SOA record requires 5 values: a serial number, a refresh time, a retry time, an expire time, and a minimum ttl (time to live). The first time the data is created, the serial number is set to 1, the refresh time is set to 3 hours, the retry time is set to 1 hour, the expire time is set to 1 week, and the minimum ttl is set to 1 day. The **-o** option changes these values except for the serial number. Each subsequent time *hosts_to_named* is run, the serial number is incremented. If any of the other fields in the SOA record are modified, the changed values are retained.

If there are files named **spcl.***DOMAIN* or **spcl.***NET* in the current directory, **$INCLUDE** directives are added to the corresponding **db.***DOMAIN* or **db.***NET* file for the **spcl** file. In this way, special data can be added to the data generated by *hosts_to_named*.

The first time *hosts_to_named* is run, it creates a default boot file for a primary name server. Each subsequent time *hosts_to_named* is run, the boot file is updated if necessary. New entries are made in the boot file for any additional networks or domains not already in the boot file. No entries are deleted from the boot file.

The boot file for a caching-only server, **boot.cacheonly**, is created if it does not exist. The boot files for secondary servers, **boot.sec.save** and **boot.sec**, are created if the **-z** or **-Z** options are used. The boot files for secondary servers are created new each time from the primary server boot file so that they are equivalent.

EXAMPLES

Create name server data for networks 15.19.8 and 15.19.9 in **div.inc.com.**

hosts_to_named -d div.inc.com -n 15.19.8-n 15.19.9

Create name server data for networks 15.19.8 and 15.19.9 in **div.inc.com.** Ignore aliases in the host table and include 2 mail hubs - **aaa.div.inc.com** and **bbb.mkt.inc.comk**. Put all of the options in a file.

hosts_to_named -f option_file

Option_file contains the following lines:

-d div.inc.com
-n 15.19.8-n 15.19.9
-m 20:aaa
-m 30:bbb.mkt.inc.com
-A

Network 15.19.15 has hosts in the **xx.inc.com** domain and the **div.inc.com** domain. Create name server data for **xx.inc.com**. Create only pointer (PTR) data for hosts in **div.inc.com** on network 15.19.15 (this requires the hosts in **div.inc.com** to have the canonical name or an alias of the form **x.div.inc.com**).

hosts_to_named -d xx.inc.com -n 15.19.15-p div.inc.com

Create name server data for network 15.19.8 in **div.inc.com**. Include **div.inc.com** data from network 15.19.15 but do not create pointer (PTR) data for 15.19.15 since that is being handled by the **xx.inc.com** server.

hosts_to_named -d div.inc.com -n 15.19.8-a 15.19.15

AUTHOR

hosts_to_named was developed by HP.

FILES

/etc/hosts	the host table
named.boot	primary server boot file
boot.cacheonly	caching only server boot file
boot.sec.save	secondary server boot file
boot.sec	secondary server boot file
db.127.0.0	pointer information for 127.0.0.1
db.cache	stub cache file for root server addresses
db.root	data for servers for the root domain
db.DOMAIN	address and other data for a domain
db.DOMAIN.in-addr	pointer data for all network-numbers
db.NET	pointer data for a network-number

SEE ALSO

named(1M), RFC1034, RFC1035

NAME

hpux - HP-UX bootstrap and installation utility

DESCRIPTION

This is a generic manual entry documenting the **hpux** HP-UX-specific initial system loader utility (see *isl*(1M)) for bootstrap and first-time installation. **hpux** is implemented in hardware-specific versions.

- For the Series 700 implementation, see *hpux_700(1M).*
- For the Series 800 implementation, see *hpux_800(1M).*

AUTHOR

hpux was developed by HP.

SEE ALSO

hpux_700(1M), hpux_800(1M).

NAME

hpux - HP-UX bootstrap utility

SYNOPSIS

hpux [*HP-UX arguments*] [**boot**] [*devicefile*] [*arguments*]
hpux ls [*devicefile*]
hpux restore *devicefile*
hpux -v

DESCRIPTION

hpux is the HP-UX specific initial system loader (*isl*(1M)) utility for bootstrap and first-time installation. It supports the following operations as indicated in the SYNOPSIS above:

hpux boot
Loads an object file from an HP-UX file system, *lif*(4), or raw device, then transfers control to the loaded image.

hpux ls Lists the contents of HP-UX directories in a format similar to *ls*(1).

hpux restore
Used for system installation and recovery.

hpux -v Lists the revision of the hpux loader.

The command sequences for all operations are presented under **SYNOPSIS** above. They can be either entered interactively to *isl*(1M) or present in an *isl*(1M) **autoexecute** file.

Numerical Representations

hpux accepts numbers (i.e., numeric constants) in many of its options. Numbers follow the C language notation for decimal, octal, and hexadecimal constants. A leading zero implies octal and a leading 0x or 0X implies hexadecimal. For example, 037, 0x1F, 0X1f, and 31 all represent the same number: decimal 31.

Devicefiles

hpux **boot, restore**, and **ls** operations accept *devicefile* specifications, which have the following format:

manager(path,n)filename

They are called *devicefiles* because they are composed of a device name part and a file name part. In the device part, *manager(path,n), manager* is the class of device manager to be used, for example **disk**. *path* specifies the physical hardware path to the device in either explicit or mnemonic form. *n* is the minor number which controls manager dependent functionality. The file name part, *filename*, is either a standard HP-UX path name or a *lif*(4) file name preceeded by :. Some *hpux* operations have defaults for particular components. A *devicefile* specification containing only a device part specifies a raw device. A *devicefile* specification containing a file name preceeded by : implies that the associated device part names a device containing a *lif*(4) file and that the named file resides on that LIF device. A *devicefile* specification containing a standard HP-UX file name implies that the associated device part names a device containing an HP-UX file system. The named file resides in that file system.

Managers

Currently, *hpux* supports the **disk, tape**, and **lan** managers. **Disk** manages all disks. **Lan** manages remote boot through the LAN connection. **Tape** manages the DDS tape drive.

Hardware Paths

The hardware path in a *devicefile* can be specified explicitly through the use of numbers, /, and ., or can be specified through mnemonics. For example, **scsi.1** specifies a device at address 1 attached to **Core** SCSI. Explicitly, this same path could be specified as **2/0/1.1**. The following table indicates the supported mnemonics.

Mnemonic	Description
scsi	Core SCSI
lan	Core LAN
eisa	EISA

Minor Numbers

The minor number, *n*, in a *devicefile* specification controls driver dependent functionality. Currently, all minor numbers should be 0 as they are not used and are reserved for future enhancements.

File Names

File names can be omitted in which case the device is accessed as if there is no file system format on the device. Specified file names can be standard HP-UX path names or a *lif*(4) file name preceeded by :.

Standard HP-UX files can be *CDF* files and symbolic links. In the case of symbolic links, the target of the link must reside in the same physical file system as the link itself. No attempt to trace links across file systems or physical disks is done.

DEFAULTS

Currently, only the hardware path, minor number, and *CDF* context have defaults.

Hardware Paths

When the hardware path element is not specified, *hpux* obtains a default for it from information maintained by *pdc*(1M). The path will be that used to load and run *isl*(1M).

Minor Numbers

Currently, minor numbers are not used but are reserved for future enhancements. They default to 0.

CDF Context

If the target of an *hpux* command is a *CDF* and context is not specified, the contexts are searched in the following order.

cnode name **HP-PA localroot default**

cnode name is extracted from **/etc/clusterconf**. The context name can be entered explicitly in the following form

<file>+/<context>

BOOT

The **boot** operation loads an object file from an HP-UX file system, *lif*(4), or raw device as specified by the *devicefile*. It then transfers control to the loaded image, passing optional arguments.

Some missing components in the *devicefile* are supplied with a default (see DEFAULTS). For example, a *devicefile* of **disk(;)/vmunix.new** would actually yield **disk(scsi.1;0)/vmunix.new** and a *devicefile* of **disk(scsi.6;)/hp-ux**, for booting from the disk at Core SCSI address 6, would yield **disk(scsi.6;0)/hp-ux**.

The boot operation is the default command for *hpux*. And so,

hpux

and

hpux disk(;0)/hp-ux

are both valid and equivalent when booting from disk. However, if the boot command is specified, it must be given a *devicefile*. Therefore,

hpux boot

is not valid.

Once the image has been loaded, **boot** gives the sizes of the **TEXT**, **DATA**, and **BSS** segments and the entry offset, before transferring control to it.

The **boot** operation accepts several optional arguments. These arguments are passed along to the loaded image. If the loaded image is an HP-UX operating system kernel, the following arguments are available:

-f*number* This option takes a number (see Numerical Representations) and passes it as the flags word to the kernel.

-i*string* This option accepts a string that specifies the initial *run-level* for *init*(1M). Note that the *run-level* specified overrides any *run-level* specified in an **initdefault** entry in **/etc/inittab** (see *inittab*(4)).

These optional arguments are specified before the **boot** command. However, if the image is not the HP-UX operating system, up to eight optional arguments of any type can be passed to the image. *hpux* does no interpretation of these arguments before passing them to the image. These arguments are specified after *devicefile* and are mutually exlusive to the HP-UX arguments.

boot currently places some minor restrictions on object files it can load. It accepts only the HP-UX magic numbers **SHAREMAGIC** (0410), **EXECMAGIC** (0407), and **DEMANDMAGIC** (0413) (see *magic*(4)). The object file must contain an Auxiliary Header of the **HPUX_AUX_ID** type and it must be the first Auxiliary Header (see *a.out*(4)).

ls

The **ls** operation lists the contents of the HP-UX directory specified by the optional *devicefile*. The output is similar to that of the **ls -alFH** command, except that the owner, group, and date information is not printed.

RESTORE

The **restore** operation is provided as a recovery mechanism in the event that a disk becomes totally corrupted. It copies data from a properly formatted bootable tape to disk. When this tape contains a back-up image of the disk, the entire disk is restored. To create a properly formatted tape (DDS ONLY), the following commands should be executed:

dd if=/usr/lib/uxbootlf.700 of=/dev/rmt/0mn bs=2k
dd if=/dev/rdsk/1ss of=/dev/rmt/0m bs=64k

The first *dd*(1) puts a boot area on the tape, making it a bootable image. Once the boot image is on tape, the tape is *not* rewound. The next *dd* appends an image of the disk to the tape. This whole process takes about one hour for a 660Mb HP2213 disk. To avoid later problems with *fsck*(1) after the disk is restored, bring the system to single-user mode and type **sync** a few times before doing the second *dd*. Once created, the tape can be used to completely restore the disk:

1. Insert the tape into the tape drive.
2. Instruct the machine to boot to ISL from the tape.

This is usually done by specifying **scsi.3** as the boot path.

3. Enter the following in response to the ISL prompt:

ISL> hpux restore disk(scsi.1;0)

This restores the disk image from the tape to the actual disk at **scsi.1**. *ANY EXISTING DATA ON THE DISK WILL BE LOST*. The restoration process also takes about one hour for a 660Mb drive.

WARNING: This command destroys the contents of the device specified by *devicefile*. Also, this command may be replaced in the future by superior installation and recovery mechanisms. At that time, this command will be removed.

EXAMPLES

As a preface to the examples which follow, here is a brief overview of HP-UX system boot-up sequences.

Automatic boot processes on various HP-UX systems follow similar general sequences. When power is applied to the HP-UX system processor, or the system **Reset** button is pressed, processor-dependent code (firmware) is executed to verify hardware and general system integrity (see *pdc*(1M)). After checking the hardware, *pdc* announces that the user has the option to override the **autoboot** sequence by pressing the **Esc** key. At that point, a message resembling the following usually appears on the console.

(c) Copyright. Hewlett-Packard Company. 1991.
All rights reserved.

PDC ROM rev. 130.0
32 MB of memory configured and tested.

Selecting a system to boot.
To stop selection process, press and hold the ESCAPE key...

If no keyboard activity is detected, *pdc* commences the **autoboot** sequence by loading *isl* (see *isl*(1M)) and transferring control to it. Since an **autoboot** sequence is occurring, *isl* merely announces itself, finds and executes the **autoexecute** file which, on an HP-UX system, requests that *hpux* be run with appropriate arguments. Messages similar to the following are displayed on the console:

Booting from: scsi.6 HP 2213A
Hard booted.
ISL Revision A.00.09 March 27, 1990
ISL booting hpux boot disk(;0)/hp-ux

hpux then announces the operation it is performing, in this case **boot**, the *devicefile* from which the load image comes, and the **TEXT** size, **DATA** size, **BSS** size, and start address of the load image. The following is displayed before control is passed to the image.

```
Secondary Loader 9000/700
Revision 1.1
Booting disk(scsi.6;0x0)/hp-ux
966616+397312+409688 start 0x6c50
```

The loaded image, in this case an HP-UX operating system kernel, then starts by giving numerous configuration and status messages.

In order to use *hpux* interactively, *isl* must be brought up in interactive mode by pressing the **Esc** key during the interval allowed by *pdc*(1M). *pdc* then searches for and displays all bootable devices then presents a set of boot options. If the appropriate option is chosen, *pdc* loads *isl*(1M) and *isl* interactively prompts for commands. Information similar to the following is displayed:

```
Selection process stopped.

Searching for Potential Boot Devices.
To terminate search, press and hold the ESCAPE key.

Device Selection    Device Path          Device Type
------------------------------------------------------------

P0                  scsi.6.0             QUANTUM PD210S
P1                  scsi.1.0             HP    2213A
p2                  lan.ffffff-ffffff.f.f hpfoobar

b) Boot from specified device
s) Search for bootable devices
a) Enter Boot Administration mode
x) Exit and continue boot sequence

Select from menu: b p0 isl

Trying scsi.6.0
Boot path initialized.
Attempting to load IPL.

Hard booted.

ISL Revision A.00.09 March 27, 1990

ISL>
```

Although all the operations and options of *hpux* can be used from *isl* interactively, they can also be executed from an **autoexecute** file. In the examples below, all user input is in boldface type.

Boot

```
ISL> hpux boot disk(scsi.0;0)/hp-ux
Secondary Loader 9000/700
Revision 1.1
Booting disk(scsi.0;0)/hp-ux
966616+397312+409688 start 0x6c50
```

If *ISL* has been booted from the disk at **scsi.0**, the command

```
hpux
```

would be equivalent to the example.

Booting a CDF

```
ISL> hpux boot disk(scsi.0;0)/hp-ux+/localroot
Secondary Loader 9000/700
```

```
Revision 1.1
Booting disk(scsi.0;0)/hp-ux+/localroot
966616+397312+409688 start 0x6c50
```

In this example, the context *localroot* is booted. If it were omitted *hpux* would try each context from the list specified in the *defaults* section of this man page until finding a match. That context would then be booted.

Booting Another Kernel

```
ISL> hpux boot disk(scsi.6;0)/vmunix.new
Secondary Loader 9000/700
Revision 1.1
Booting disk(scsi.6;0)/vmunix.new
966616+397312+409688 start 0x6c50
```

Here *hpux* initiates a **boot** operation where the name of the object file is **vmunix.new**.

Booting from LAN

```
ISL> hpux boot lan(;0)hp-ux
Secondary Loader 9000/700
Revision 1.1
Booting lan(;0)hp-ux
966616+397312+409688 start 0x6c50
```

This example shows how to boot a diskless client from the LAN. It assumes that ISL was also booted from the LAN. The file name must be specified and be no more than 127 characters in length. Booting to *isl* from a local disk and then requesting an image to be loaded from the LAN is *not* supported.

If *hp-ux* is the target of the intended boot operation, and it is being booted from *CORE* LAN , the command

```
hpux
```

is equivalent to the above example.

Booting from a Raw Device

```
ISL> hpux boot tape(scsi.3;0)
Secondary Loader 9000/700
Revision 1.1
Booting tape(scsi.3;0)
966616+397312+409688 start 0x6c50
```

This example shows booting from a raw device (i.e., no file system is on the device). It assumes that a kernel has been put onto tape with *dd*(1) and a 2K-byte block size. Note that no file name is specified in the *devicefile*. The device is a DDS tape drive and therefore **tape** is the manager used.

Booting to Single User Mode

```
ISL> hpux -is boot disk(scsi.0;0x0)/hp-ux
Secondary Loader 9000/700
Revision 1.1
Booting disk(scsi.0;0)/hp-ux
966616+397312+409688 start 0x6c50

        Kernel Startup Messages Omitted

INIT: Overriding default level with level 's'

INIT: SINGLE USER MODE
WARNING:  YOU ARE SUPERUSER !!

#
```

In this example, the **-i** option is used to make the system come up in *run-level* **s**, for single user mode of operation.

Listing Directory Contents

```
ISL> hpux ls disk(;)/.
Secondary Loader 9000/700
Revision 1.1

drwxr—xr—x    9      2048 ./
drwxr—xr—x    6      2048 ../
drwxr—xr—x    2      4096 lost+found/
—rw—rw—r——    1       746 .profile
drwxrwxr—x    2      1024 bin/
drwxr—xr—x   12      1024 dev/
drwxrwxr—x    5      1024 etc/
drwxrwxrwx    2        64 tmp/
drwxrwxr—x    3      1024 usr/
Hrwxrwxr—x    3      1024 foo+
—rwxr—xr—x    1    884736 hp-ux*
—rwxr—xr—x    1    884736 SYSBCKUP*
—rwxr—xr—x    1   1032192 hp-ux.test*
```

The contents of the root directory (/) on the root disk are listed. The format shows the file protections, number of links, and size in bytes for each file in the directory. There are three available kernels to boot: hp-ux, hp-ux.test, and SYSBCKUP. Listing the files of a diskless server from a diskless client is not supported.

Getting the Version

```
ISL> hpux -v
Secondary Loader 9000/700
Revision 1.1
@(#) Revision 1.1 Wed Dec 10 17:24:28 PST 1986
ISL>
```

The **-v** option is used to get more detailed information about the version of *hpux*.

Installing/Recovering a System

```
ISL> hpux restore disk(scsi.6;0)

Restore scsi.6
Skipping boot area
Skip done
Copying tape
Copy done 10560
Restoration Done.
```

The **restore** command will copy a specially formatted DDS tape to disk. This is useful when copying the install image or a back-up of your disk from tape. The **restore** command displays forward progress as it positions on the tape and copies data to the disk. The final message indicates that the restore has completed and, in this case, 10560 blocks of 64k bytes has been copied. Currently, the **restore** command only supports disk devices as output.

WARNING: This command destroys the contents of the specified target device. Also, this command may be replaced in the future by superior installation and recovery mechanisms. At that time, this command will be removed.

DIAGNOSTICS

When errors occur, *hpux* prints diagnostic messages indicating the cause of the error:

bad number in minor spec
Illegal minor number in *devicefile* specification.

Unable to initialize boot device
The device specified by *devicefile* cannot be initialized. Check cables, addresses, and device status (On-line/Off-line).

No such command
Illegal or unknown command specified.

Bad fs magic
Device has a file system of unknown type or no file system.

bad number in flags spec
number argument specified with the **-f** option is invalid.

bad magic
Specified object file does not have a legal magic number.

Exec failed
An unknown error caused a failure while launching the application, or the application returned prematurely.

isl not present, please hit system RESET button to continue
An unsuccessful **boot** operation has overlaid *isl* in memory making it impossible to return control to *isl*.

short read
Specified object file is internally inconsistent. It is not long enough.

would overlay
Loading the specified object file would overlay *hpux*.

BTLAN: Using /hp-ux instead
Encountered a boot request by a cluster client asking for a filename more than seven characters in length. *hpux* cannot supply that file and is launching *hp-ux* instead.

Unable to open lif file
The *lif*(4) file to be booted is not bootable or does not exist.

bad number in path spec
A number or mnemonic specified as the path is invalid.

iodc_open failure in fopen
devicefile specifies a path that cannot be opened. See **Unable to initialize boot device .**

SEE ALSO
boot(1M), fsck(1M), init(1M), isl(1M), pdc(1M), errno(2), a.out(4), inittab(4), magic(4).

NAME

hpux - HP-UX bootstrap and installation utility

SYNOPSIS

hpux [**-o**] [**-F**] [**-m** [**p** | **s** | **x**]] [**-a** [**C** | **R** | **S** | **D**] *devicefile*] [**-f***number*] [**-i***string*] [**boot**] [*devicefile*]
hpux install [**from** *devicefile*] [**to** *devicefile*]
hpux ls [*devicefile*]
hpux set autofile *devicefile string*
hpux show autofile [*devicefile*]
hpux -v

Obsolete:
hpux copy devicefile devicefile

DESCRIPTION

hpux is the HP-UX-specific initial system loader (*isl*(1M)) utility for bootstrap and first-time installation. It supports the following operations as indicated in the SYNOPSIS section above:

boot	Loads an object file from an HP-UX file system or raw device and transfers control to the loaded image.
install	Used during first time installation. Attempts to execute an install image from a properly formatted installation device.
ls	Lists the contents of HP-UX directories in a format similar to *ls*(1).
show autofile	Displays the contents of the **autoexecute** file.
set autofile	Changes the contents of the **autoexecute** file to that specified by *string*.
-v	Display the release and version numbers of the *hpux* utility.
copy	Obsolete. May not work on your system. Copies data between HP-UX files and/or raw devices. This operation is not recommended because it may damage your file systems.

hpux commands can be given interactively from the keyboard, or provided in an *isl autoexecute* file.

NUMBERS

hpux accepts numbers (i.e. numeric constants) in many of its options. Numbers follow the *C* language notation for decimal, octal, and hexadecimal constants. A leading 0 (zero) implies octal and a leading 0x or 0X implies hexadecimal. For example, 037, 0x1F, 0X1f, and 31 all represent the same number: decimal 31.

DEVICEFILES

hpux **boot**, **install**, **ls**, and **copy** operations accept *devicefile* specifications, which have the following format:

manager(w/x.y.z;n,s)filename

They are called *devicefiles* because they are composed of a device name part and a file name part. In the device part, *manager(w/x.y.z;n,s)*, *manager* is the name of an HP 9000 Series 800 I/O System manager (i.e. device driver) such as **disc0**. *W/x.y.z* is the physical hardware path to the device, identifying bus converters, slot numbers, and hardware addresses. (Bus converter specifications are necessary only for models with bus converters such as the Model 850. *hpux* only allows one level of bus converters). *N* is the minor number which controls manager dependent functionality. *S* is the file skip count. For devices, this parameter describes how many files must be skipped (from the beginning of the tape) before the desired file can be accessed. It has a default value of **0**, and is completely ignored for other devices. The file name part, *filename*, is a standard HP-UX path name. Some *hpux* operations have defaults for particular components. A *devicefile* specification containing a device part only specifies a raw device. A *devicefile* specification containing a file name implies that the associated device part names a device containing an HP-UX file system. The named file resides in that file system. For example, a typical boot *devicefile* specification is disc0(2/4.0.0;0)hp-ux.

Managers

Currently, *hpux* supports the **disc0**, **disc1**, **disc2**, **disc3**, **disc30**, **tape1**, **tape2**, and **lan1** managers. **Disc0** manages all CS/80 disks connected via HP-IB, including cartridge tape devices, and **disc2** manages all CS/80 disks connected via the HP27111 interface. **disc1** manages all CS/80 disks connected via NIO HP-IB, including cartridge tape devices. **disc3** manages all disks connected via SCSI, including cartridge tape

devices, and **disc30** manages all *auto-changer* type disk devices. **lan1** manages remote boot through the HP28652A NIO based LAN interface. Remote boot is currently supported on this card only and not on any CIO- based LAN card. **tape1** manages the HP7974, HP7978, and HP7980 tape drives via HP-IB, and **tape2** manages tape drives via SCSI.

Hardware Paths
The hardware path in a *devicefile* specification is an arbitrary-length string of numbers, each suffixed by slash, (/), followed by an arbitrary-length string of numbers separated by periods (.). Each number identifies a hardware component. Hardware components suffixed by slashes indicate bus converters and may not be necessary on your machine. A single number is the shortest path specification. In *w/x.y.z* above, *w* would be the bus converter number, *x* would be the MID-BUS module number, *y* would be the CIO slot number, and *z* would be the HP-IB address or HP27111 bus address.

Minor Numbers
The minor number, *n*, in a *devicefile* specification controls driver-dependent functionality. The *HP-UX System Administrator* manuals describe specific minor number encodings for individual drivers. Since *hpux* manages its own logical units, it consequently ignores any logical unit information that may be specified in the minor number field of a *devicefile*. For more information on the minor number formats for **disc0**, **disc1**, **disc2**, **disc3**, **disc30**, **tape1**, **tape2**, **lan1**, refer to the *HP-UX System Administrator* manuals for your system.

Skip Counts
The skip count, *s*, in a *devicefile* specification controls how many files must be skipped before the desired file is reached. It is relative to the beginning of the tape, and is defined only for **tape1** and **tape2** devices. It is ignored for all others. If not specified, **0** is assumed.

File Names
File names are standard HP-UX path names. No preceding slash (/) is necessary and specifying one will cause no problems. File names are not root (i.e. /) relative. For example, with **disc0**, **disc1**, **disc2**, and **disc3**, they are relative to the section specified in the minor number (in the device part) of the *devicefile* specification.

Context-Dependent Files
A file name can specify a context-dependent file (see *cdf*(4)). This is done in either of two ways. If the file name is of the form,

filename+/*context*

the operation is performed on *context* within the file specified by *filename*. If the path name is of the form,

filename

and the file is a *CDF*, a default context is chosen for the operation (see **DEFAULTS**).

DEFAULTS
Default values chosen by *hpux* to complete a command are obtained through a sequence of steps. First, any components of the command specified explicitly are used. If the command is not complete, *hpux* attempts to construct defaults from information maintained by *pdc*(1M). If sufficient information to complete the command is unavailable, the **autoexecute** and **clusterconf** files are searched. On a search failure for any reason, any remaining unresolved components of the command are satisfied by hard-coded defaults. The following sections detail how specific components will default.

Managers
The choice for a default manager is determined exactly as stated above. However, there is no hard-coded default. If none can be chosen, this is an error.

Hardware Paths
When the hardware path element is not specified, *hpux* obtains a default for it from information maintained by *pdc*. Like the default for the manager element, the hardware path element has no hard-coded default.

Minor Numbers
If the minor number element is not supplied, a default will be taken from the *autoexecute* file since *pdc* does not supply any minor number information. Failing that, the hard-coded default of 0 is used.

Skip Counts
When the skip count is not specified, the hard-coded default of 0 is always used. Other sources providing a

default value for this element are ignored.

File Names

For the **boot** command, a *devicefile* specification without a file name element indicates that the boot device does not contain an HP-UX file system. It is thus interpreted as a *NULL* file name instead of a missing file name. And so, the file name is not defaulted. A completely missing *devicefile* specification does imply a missing file name and will be defaulted as usual. Since *pdc* maintains no information about file names, either the *autoexecute* file contents or the hard-coded default of **hp-ux** is chosen.

For the **ls** command, the default file name taken from the *autoexecute* file is ignored. The default value is always the hard-coded value, . (period).

For the **install** command, only components of the **from** *devicefile* can be defaulted.

Context-Dependent Files

For the **boot** command, if the file is a *CDF*, the *autoexecute* file is searched for the default context. Failing that, the **/etc/clusterconf** (see **clusterconf**(4)) file is searched for the node name which is then used as the context. Finally, if all else fails, the hard-coded default, **localroot**, is chosen.

For the **ls** command, the defaults supplied by the *autoexecute* and **/etc/clusterconf** files are ignored. The hard-coded value, **localroot**, is also ignored. Contents of a CDF can be displayed by specifying the file name with an appended **+**.

boot OPERATION

The **boot** operation loads an object file from an HP-UX file system or raw device as specified by the optional *devicefile*. It then transfers control to the loaded image.

Any missing components in a specified *devicefile* are supplied with a default. For example, a *devicefile* of **vmunix.new** would actually yield **disc0(8.0.0;0)vmunix.new** and a *devicefile* of **(8.0.1)hp-ux**, for booting from the disk at HP-IB address 1, would yield **disc0(8.0.1;0)hp-ux**. Regardless of how incomplete the specified *devicefile* may be, **boot** announces the complete *devicefile* specification used to find the object file. Along with this information, **boot** gives the sizes of the **TEXT**, **DATA**, and **BSS**, segments and the entry offset of the loaded image, before transferring control to it.

The **boot** operation accepts several options. Their meanings are:

-a[**C** | **R** | **S** | **D**] *devicefile*
This option takes a devicefile specification (see DEVICEFILES) and passes it to the loaded image. If that image is an HP-UX kernel, the kernel will erase its predefined I/O configuration, and configure in the specified devicefile. When the **C**, **R**, **S**, or **D** option is specified, the kernel configures the device as the *console*, *root*, *swap*, or *dump* device, respectively. Note that **-a** can be repeated multiple times and that its use implies the **-o** option.

-f*number* This option takes a number (see NUMBERS) and passes it as the flags word to the loaded image.

-i*string* This option accepts a string that specifies the initial *run-level* for *init*(1M). Note that the *run-level* specified will override any *run-level* specified in an *initdefault* entry in **/etc/inittab** (see *inittab*(4)).

-o This option passes the console and boot device paths and drivers to the loaded image. If that image is an HP-UX kernel, the kernel will erase its predefined I/O configuration, and replace its console and root device paths and drivers with those passed from **boot**. In addition, the primary swap device is also placed on the boot device. This is useful for forcing a boot if the kernel has an incorrect I/O configuration.

-m[**p**|**s**|**x**]
If the loaded image is an HP-UX kernel, this option controls the kernel's choice of which section in a mirrored root should be ONLINE. Without this option, the kernel chooses the section that was ONLINE when the system went down, or the primary section if both sections were ONLINE. **-m** alone causes the kernel to use the previously OFFLINE section, or the secondary section if both sections were previously ONLINE. **-mp** and **-ms** specify the primary and secondary sections, respectively.

If one of the mirrored disks is inaccessible, the kernel on rare occasions cannot tell if the accessible disk was ONLINE when the system went down. Normally, it prints a message asking for help, and halts, since an incorrect choice can cause data corruption. With **-mx**, the kernel always uses the accessible disk, regardless of the possibility of corruption; it is useful for the AUTO file on systems where availability is more important than reliability (for example, a machine controller, or a system that provides access to a read-only database of which there are other copies).

-F This option is for use with SWITCHOVER/UX software, which is not in the HP-UX core. It tells the kernel to ignore any locks on the boot disk, and should only be used when it is known that the processor with the lock is no longer running. Without this option, if a disk is locked by another processor, the kernel refuses to boot from it, to avoid the corruption that would result if the other processor were still using the disk.

boot currently places some minor restrictions on object files it can load. It accepts only the HP-UX magic numbers **SHAREMAGIC** (0410) and **DEMANDMAGIC** (0413) (see *magic*(4)). The object file must contain an Auxiliary Header of the **HPUX_AUX_ID** type and it must be the first Auxiliary Header (see *a.out*(4)).

install OPERATION

The **install** operation is used during first-time installation to load and execute an install image from a properly formatted installation device. The **install** operation is automatically translated to a **boot** operation from a *devicefile* based on the model of your system. The path to the install source device is usually the same as the one specified to *pdc* but may be selected by the **from** option.

set autofile OPERATION

The **set autofile** operation overwrites the contents of the *autoexecute* file, *autofile*, with the string specified (see *autoexecute* in EXAMPLES section).

show autofile OPERATION

The **show autofile** operation prints the contents of the *autoexecute* file, *autofile* , on the display (see *autoexecute* in EXAMPLES section).

copy OPERATION

The **copy** operation is obsolete and should not be used. It was originally used during first-time installation to copy installation images from one device to another. Using this operation may damage your file system.

ls OPERATION

The **ls** operation lists the contents of the HP-UX directory specified by the optional *devicefile*. The output is similar to that of the **ls -alFH** command, except that the owner, group, and date information is not printed.

The default *devicefile* for **ls** is generated in the same way as the *devicefile* for **boot**. However, the default file name is ".".

EXAMPLES

Before going over specific examples of the various options and operations of *hpux*, here is an outline of the steps taken in the automatic boot process. Although the hardware configuration and boot paths shown are for a single Series 800 machine, the user interfaces are consistent across all models. When the system **RESET** button is depressed, *pdc* executes self-test, and assuming the hardware passes, *pdc* announces itself, issues a **BELL**, and gives the user 10 seconds to override the *autoboot* sequence, by entering any character. The following is typically displayed on the console.

```
Processor Dependent Code (PDC) revision 2

Console path = 8.1.0.0.0.0.0
Primary boot path = 8.0.0.0.0.0.0
Alternate boot path = 8.2.3.0.0.0.0

Autoboot from primary boot path enabled. To override, press any key within 10 seconds.
```

If no character is entered within 10 seconds, *pdc* commences the *autoboot* sequence by loading *isl* and transferring control to it. Because an *autoboot* sequence is occurring, *isl* merely announces itself, finds and executes the *autoexecute* file which, on an HP-UX system, requests that *hpux* be run with appropriate arguments. The following is displayed on the console.

```
10 seconds expired.

Booting.

Console IO Dependent Code (IODC) revision 1
Boot IO Dependent Code (IODC) revision 1

Booted.

ISL Revision 2634  August, 1986

ISL booting  hpux
```

Next *hpux* announces the operation it is performing, in this case **boot**, the *devicefile* from which the load image comes, and the **TEXT** size, **DATA** size, **BSS** size, and start address of the load image. The following is displayed before control is passed to the image.

```
Boot
: disc0(8.0.0;0x0)hp-ux
966616+397312+409688 start 0x6c50
```

Lastly, the loaded image, in this case an HP-UX operating system kernel, starts by giving numerous configuration and status messages. The system in the following example eventually comes to *init run-level* 2 for multi-user mode of operation.

```
Beginning I/O System Configuration.
cio_ca0 address = 8
    hpib0 address = 0
        disc0 lu = 0 address = 0
        disc0 lu = 1 address = 1
        disc0 lu = 2 address = 2
        disc0 lu = 3 address = 3
    mux0 lu = 0 address = 1
        .
        .
More deleted for brevity
        .
        .
graph0 lu 0 address 12
I/O System Configuration complete.
Configure called
@(#)9245XA HP-UX (sys.A.B1.10/S800) #1: Wed Dec 10 17:24:28 PST 1986
real mem = 8386560
lockable mem = 3297280
avail mem = 5197824
using 204 buffers containing 837632 bytes of memory
```

In order to use the operations and options of *hpux*, *isl* must be brought up in interactive mode. To do this simply enter a character during the 10-second interval allowed by *pdc*(1M). *pdc* then asks if the primary boot path is acceptable. Answering yes (**Y**) is usually appropriate. *pdc* then loads *isl* and *isl* interactively prompts for commands. The following is displayed.

```
Boot from primary boot path (Y or N)?> Y

Booting.

Console IO Dependent Code (IODC) revision 1
Boot IO Dependent Code (IODC) revision 1

Booted.

ISL Revision 2634  August, 1986

ISL>
```

Although all the operations and options of *hpux* can be used from *isl* interactively, they can also be executed from an *autoexecute* file. In the examples below, all user input is in boldface type.

Default Boot

ISL> **hpux**

Boot
: disc0(8.0.0;0x4)hp-ux
966616+397312+409688 start 0x6c50

Entering **hpux** initiates the default boot sequence. The boot path read from *pdc* is **8.0.0**, the manager associated with the device at that path is **disc0**, the minor number, in this case derived from the *autoexecute* file, is **4** specifying section 4 of the disk, and the object file name is **hp-ux**.

Booting Another Kernel

ISL> **hpux vmunix.new**

Boot
: disc0(8.0.0;0x0)vmunix.new
966616+397312+409688 start 0x6c50

Here *hpux* initiates a **boot** operation where the name of the object file is **vmunix.new**.

Booting from Another Section

ISL> **hpux (;3)sys.azure/S800/vmunix**

Boot
: disc0(8.0.0;0x3)sys.azure/S800/vmunix
966616+397312+409688 start 0x6c50

In this example, a kernel is booted from another section of the root disk. For example, let's say that kernel development takes place under **/mnt/azure/root.port** which happens to reside in its own section, section 3 of the root (i.e. default boot) disk. By specifying a minor number of **3**, in the above example, the object file **sys.azure/S800/vmunix** is loaded from **/mnt/azure/root.port**.

Booting from Cartridge Tape

ISL> **hpux (;4194336)hp-ux**

Boot
: disc0(8.0.0;0x400020)hp-ux
966616+397312+409688 start 0x6c50

In this example, the default boot device is an HP7914 disk with a cartridge tape at unit 1. The minor number has the cartridge tape flag set and specifies unit 1, section 0 of the device. Although the minor number was entered in decimal format, the hexadecimal form would be accepted. Since a file name is specified, it is assumed that section 0 contains a file system.

Booting from Another Disk

ISL> **hpux (8.0.1)hp-ux**

Boot
: disc0(8.0.1;0x0)hp-ux
966616+397312+409688 start 0x6c50

In this example, only the hardware path and file name are specified. All other values are boot defaults. The object file comes from the file system in section 0 of the disk, at HP-IB address 1.

Booting from LAN

ISL> **hpux lan1(32)hp-ux**

Boot
: lan1(32;0x0)hp-ux
966616+397312+409688 start 0x6c50

This example shows how to boot a cluster client from the LAN. Though this example specifies a devicefile, using default boot, shown in a previous example, is also possible. For a boot operation other than default boot, the file name must be specified and be no more than 11 characters in length. Booting to *isl* from a local disk and then requesting an image to be loaded from the LAN is **NOT** supported.

Booting from a Raw Device

ISL> **hpux tape1(8.2.3;0xa0000,1)**

```
Boot
: tape1(8.2.3;0xa0000,1)
rewinding tape1(8.2.3;0xa0000,1) ... done
skipping 1 file ... done
966616+397312+409688 start 0x6c50
```

This example shows booting from a raw device (i.e. no file system is on the device). Note that no file name is specified in the *devicefile*. The device is an HP 7974 tape drive and therefore **tape1** is the manager used. The tape drive is at CIO slot 2, HP-IB address 3. The first file on the tape will be skipped. The minor number specifies a tape density of 1600 BPI and no rewind on close. Note that, depending on the minor number, **tape1** requires the tape be written with 512 or 1024 byte blocks.

Booting to Single User Mode

ISL> **hpux -is**

```
Boot
: disc0(8.0.0;0x0)hp-ux
966616+397312+409688 start 0x6c50

Kernel Startup Messages Omitted

INIT: Overriding default level with level 's'

INIT: SINGLE USER MODE
WARNING:  YOU ARE SUPERUSER !!

#
```

In this example, the **-i** option is used to make the system come up in *run-level* **s**, for single user mode of operation.

Booting with a Modified I/O Configuration

ISL> **hpux -aC mux0(8.1) -a tape1(8.2.0)**

```
Boot
: disc0(8.0.0;0x0)hp-ux
: Adding mux0(8.1;0x0)...
: Adding tape1(8.2.0;0x0)...
966616+397312+409688 start 0x6c50
Beginning I/O System Configuration.
cio_ca0 address = 8
  hpib0 address = 0
    disc0 lu = 0 address = 0
  mux0 lu = 0 address = 1
  hpib0 address = 2
    tape1 lu = 0 address = 0
I/O System Configuration complete.

        Additional Kernel Startup Messages Omitted
```

Here a tape driver is configured in at CIO slot 2, HP-IB address 0. Regardless of what was present in the kernel's original I/O configuration, the driver **tape1** is now configured at that hardware path. Similarly, **mux0** is configured in at CIO slot 1 which is to be the console. The only other devices configured are the console and root device, which **boot** derived from *pdc* .

First-Time Installation

ISL> **hpux install to disc0(8.0.0)**

```
Boot
: tape1(8.2.3;0xa0000,1)
: Adding disc0(8.0.0;0x0)...
rewinding tape1(8.2.3;0xa0000,1) ... done
skipping 1 file ... done
966616+397312+409688 start 0x6c50
```

This is an example of the **install** operation on a model 840. In this example, *pdc* was instructed to boot from the boot path, "8.2.3". *hpux* translated the **install** operation to a **boot** operation from this boot path using defaults characteristic of the model 840. As a general rule, smaller systems (for example, model 815) default to installing from cartridge tape while larger systems (such as, model 850) default to reel tape.

In this case, the optional **to** argument is used to force an installation to the HP-IB disk at "8.0.0". *hpux* translates this option to an equivalent **-a** construct, which is necessary for installation. Though not shown by this example, the source installation device could have been changed by specifying the **from** *devicefile* argument. Any missing components of the *devicefile* associated with the **from** argument will be defaulted. However, those associated with the **to** argument will not. The command in this example is functionally equivalent to the following command on a model 840.

ISL> **hpux -a disc0(8.0.0) tape1(8.2.3;0xa0000,1)**

Note: Due to variety of tasks being performed and the slow nature of tape based boot devices, this operation my take a good deal of time before announcing completion. Forward progress can be monitored by watching the tape or seven segment display on your machine.

Displaying the autoexecute file

ISL> **hpux show autofile**
Show autofile
: AUTO file contains (hpux -i2)

In this example, **show autofile** is used to print the contents of the *autoexecute* file residing in the boot area, section 6, on the device from which *hpux* was booted. Optionally, a *devicefile* may be specified in order to read the *autoexecute* file from the boot area of some other boot device.

Changing the autoexecute file

ISL> **hpux set autofile (;6) "hpux (;4)hp-ux.std"**
Set autofile
: AUTO file now contains "hpux (;4)hp-ux.std"

This example shows how to change the contents of the *autoexecute* file. Once done, the system may be reset and the new command will be used during any unattended boot.

Listing Directory Contents

ISL> **hpux ls**

```
Ls : disc0(8.0.0;0x4).
drwxr-xr-x    9       2048 ./
drwxr-xr-x    6       2048 ../
drwxr-xr-x    2       4096 lost+found/
-rw-rw-r--    1        746 .profile
drwxrwxr-x    2       1024 bin/
drwxr-xr-x   12       1024 dev/
drwxrwxr-x    5       1024 etc/
drwxrwxrwx    2         64 tmp/
drwxrwxr-x    3       1024 usr/
Hrwxrwxr-x    3       1024 foo+
-rwxr-xr-x    1     884736 hp-ux*
-rwxr-xr-x    1     884736 SYSBCKUP*
-rwxr-xr-x    1    1032192 hp-ux.test*
```

The contents of the root directory (/)**on**the**root** The format shows the file protections, number of links, and size in bytes for each file in the directory. There are three available kernels to boot: **hp-ux**, **hp-ux.test**, and **SYSBCKUP**. Listing the files of a cluster server from a cluster client is not supported. Listing files within a *CDF* may be done by appending a "+" to the *CDF* file name. For example, ISL> **hpux ls foo+** will list the contents of the **foo** *CDF*.

Getting the Version

ISL> **hpux -v**

```
Release: 1.1
Release Version:
@(#)9245XA HP-UX (sys.A.B1.10/HPUXBOOT) #1: Wed Dec 10 17:24:28 PST 1986
```

The **-v** option is used to get the version numbers of *hpux*.

DIAGNOSTICS

In the instance of an error **hpux** prints diagnostic messages which indicate the cause of the error. These messages may be grouped General, Boot, Copy, Configuration, and System Call. A description of the System Call error messages may be found in *errno*(2). The remaining messages are described below.

General

bad minor number in devicefile spec
The minor number in the *devicefile* specification is illegal.

bad path in devicefile spec
The hardware path in the *devicefile* specification is illegal.

command too complex for parsing
The command line contains too many arguments.

no path in devicefile spec
The *devicefile* specification does not contain a hardware path component and must.

panic (in hpuxboot): (display==*number***, flags==***number***)** *string*
A severe internal *hpux* error has occurred. Report to your nearest HP Field Representative.

Boot

bad magic
The specified object file does not have a legal magic number.

bad number in flags spec
The flags specification in the **-f** option is illegal.

booting from raw character device
In booting from a raw device, the manager specified only has a character interface. This may cause problems if the block size is incorrect.

Isl **not present, please hit system RESET button to continue**
An unsuccessful **boot** operation has overlaid *isl* in memory. It is impossible to return control to *isl*.

short read
The specified object file is internally inconsistent, it is not long enough.

would overlay
Loading the specified object file would overlay *hpux*.

Copy

cannot open destination device/file
The destination device or file could not be opened for writing.

cannot open source device/file
The source device or file could not be opened for reading.

fchmod failure (warning only)
The access mode of the destination file could not be changed.

fchown failure (warning only)
The owner and/or group of the destination file could not be changed.

fstat failure (warning only)
One or more of the owner, group, or mode of the source file could not be determined. The default values of owner and group are **0** and **0**. The default mode is **0777**.

read failure
An error was encountered reading from the source device or file.

umount failure on destination device
The destination device could not be dismounted. Its file system may have been damaged as a result. *fsck* should be run before mounting the file system.

umount failure on source device
The source device could not be dismounted. Since it was mounted read-only, the integrity of its file

system is not at risk.

write failure
An error was encountered writing to the destination device or file.

Configuration

cannot add path, error *number*
An unknown error has occurred in adding the hardware path to the I/O tree. The internal error number is given. Contact your HP Field Representative.

driver does not exist
The manager specified is not configured into *hpux*.

driver is not a logical device manager
The manager name given is not that of a logical device manager and cannot be used for direct I/O operations.

error rewinding device
An error was encountered attempting to rewind a device.

error skipping file
An error was encountered attempting to forward-space a tape device.

negative skip count
The skip count, if specified, must be greater than or equal to zero.

no major number
The specified manager has no entry in the block or character device switch tables.

path incompatible with another path
Multiple incompatible hardware paths have been specified.

path long
The hardware path specified contains too many components for the specified manager.

path short
The hardware path specified contains too few components for the specified manager.

table full
Too many devices have been specified to *hpux*.

SEE ALSO
boot(1M), fsck(1M), init(1M), isl(1M), pdc(1M), errno(2), a.out(4), inittab(4), magic(4).

NAME

ifconfig - configure network interface parameters

SYNOPSIS

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

DESCRIPTION

ifconfig is used to assign an address to a network interface and/or configure network interface parameters. **ifconfig** must be used at boot time to define the network address of each interface present on a machine. It can also be used at other times to redefine an interface's address or other operating parameters.

Command-Line Arguments

interface A string of the form *name unit*, such as **lan0**. (See DEPENDENCIES.)

address_family Name of protocol on which naming scheme is based. An interface can receive transmissions in differing protocols, each of which may require separate naming schemes. Therefore, it is necessary to specify the *address_family*, which may affect interpretation of the remaining parameters on the command line. The only address family currently supported is **inet** (DARPA-Internet family).

address Either a host name present in the host name database (see *hosts*(4)), or a DARPA Internet address expressed in Internet standard "dot notation". The host number can be omitted on 10-Mbyte/second Ethernet interfaces (which use the hardware physical address), and on interfaces other than the first.

dest_address Address of destination system. Consists of either a host name present in the host name database (see *hosts*(4)), or a DARPA Internet address expressed in Internet standard "dot notation".

parameters The following operating parameters can be specified:

up Mark an interface "up". Enables interface after an "ifconfig down." Occurs automatically when setting the address on an interface. Setting this flag has no effect if the hardware is "down".

down Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well. This action does not automatically disable routes using the interface.

trailers Request the use of a "trailer" link-level encapsulation when sending. If a network interface supports **trailers**, the system will, when possible, encapsulate outgoing messages in a manner that minimizes the number of memory-to-memory copy operations performed by the receiver. On networks that support Address Resolution Protocol, 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 (see NOTES).

-trailers Disable the use of a "trailer" link-level encapsulation (default).

arp Enable the use of Address Resolution Protocol in mapping between network level addresses and link-level addresses (default). This is currently implemented for mapping between DARPA Internet addresses and 10-Mbyte/second Ethernet addresses.

-arp Disable the use of Address Resolution Protocol.

metric *n* Set the routing metric of the interface to *n*, default 0. The routing metric is used by the routing protocol (see *gated*(1m)). Higher metrics have the effect of making a route less favorable; metrics are counted as additional hops to the destination network or host.

debug Enable driver-dependent debugging code. This usually turns on extra console error logging.

-debug Disable driver-dependent debugging code.

netmask *mask*
(Inet only) Specify how much of the address to reserve for subdividing networks into sub-networks. *mask* includes the network part of the local address, and the subnet part which is taken from the host field of the address. *mask* can be specified as a single hexadecimal number with a leading **0x**, with a dot-notation Internet address, or with a pseudo-network name listed in the network table (see *networks*(4)). *mask* contains 1's for each bit position in the 32-bit address that are to be used for the network and subnet parts, and 0's for the host part. *mask* should contain at least the standard network portion, and the subnet field should be contiguous with the network portion.

broadcast (Inet only) Specify the address that represents broadcasts to the network. The default broadcast address is the address with a host part of all 1's.

ipdst (NS only) This is used to specify an Internet host that is willing to receive IP packets encapsulating NS packets bound for a remote network. In this case, an apparent point-to-point link is constructed, and the address specified is taken as the NS address and network of the destination.

The command:

ifconfig *interface*

with no optional command arguments supplied displays the current configuration for *interface*. If *address_family* is specified, **ifconfig** reports only the details specific to that address family. Only a user who has appropriate privileges can modify the configuration of a network interface.

DEPENDENCIES

The name of an interface associated with a LAN card is **lan**, and its unit is determined as follows:

Series 300/400:
The LAN card with the lowest select code is given the interface unit number **0**; the LAN card with the next higher select code is given the interface unit number **1**; and so on.

Series 700/800:
The LAN card in the lowest hardware module in the backplane is given interface unit number **0**; the LAN card in the next higher hardware module is given interface unit number **1**; and so on. When there are two or more LAN cards in a module (e.g. CIO), interface unit numbers are assigned to LAN cards in slot order before being assigned to cards in the next higher module. For example, consider a system with two LAN cards in CIO module 4 (slot 3 and slot 7) and one LAN card in CIO module 8 (slot 5). The three cards are assigned interface unit numbers **0**, **1**, and **2**, respectively.

The **lanscan** command can be used to display the name and unit number of each interface that is associated with a LAN card (see *lanscan*(1M)).

NOTES

Currently, all HP 9000 systems can receive *trailer* packets but do not send them. Setting the *trailers* flag has no effect.

DIAGNOSTICS

Messages indicating that the specified interface does not exist, the requested address is unknown, or the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

netstat(1), lanconfig(1m), lanscan(1m) hosts(4), routing(7).

(Requires Optional LAN/X.25 Software)

NAME

inetd - Internet services daemon

SYNOPSIS

/etc/inetd [**-l**]
/etc/inetd [**-c**]
/etc/inetd [**-k**]

DESCRIPTION

inetd, the Internet super-server, invokes Internet server processes as needed. It must be running before other hosts can connect to the local host through *rcp*, *remsh*, *rlogin*, *ftp*, and *telnet*. *inetd* also supports services based on the Remote Procedure Call (RPC) protocol (NFS), such as *rwalld* and *rusersd*. If RPC servers are started by *inetd*, *portmap*(1M) must be started before *inetd*.

inetd allows one daemon to invoke many servers, thus reducing load on the system. It is normally started at system boot time. Only one *inetd* can run at any given time.

inetd starts servers for both stream and datagram type services. For stream services, *inetd* listens for connection requests on Internet stream sockets. When it finds a connection on one of its sockets, it decides which service the socket corresponds to, and then forks and invokes a server for the connection, passing the connected socket to the server as **stdin** and **stdout**. It then returns to listening for connection requests.

For datagram services, *inetd* waits for activity on Internet datagram sockets. When an incoming datagram is detected, *inetd* forks and invokes a server, passing the socket to the server as **stdin** and **stdout**. It then "waits," ignoring activity on that datagram socket, until the server exits.

inetd is normally started by the script **/etc/netlinkrc** which is run by **/etc/rc** as part of boot-time initialization. Otherwise *inetd* can be started only by the super-user.

The Internet daemon and the servers it starts inherit the LANG and TZ environment variables and the umask of the process that started the *inetd*. If *inetd* is started by the super-user, it inherits the super-user's umask, and passes that umask to the servers it starts. If desired, the script **/etc/netlinkrc** can set the umask before running *inetd*.

When invoked, *inetd* reads **/etc/inetd.conf** and configures itself to support whatever services are included in that file (see *inetd.conf*(4)). *inetd* also performs a security check if the file **/usr/adm/inetd.sec** exists (see *inetd.sec*(4)). If the Internet daemon refuses a connection for security reasons, the connection is shut down. Most RPC-based services, if their first connection is refused, attempt to connect four more times at 5-second intervals before timing out. In such cases, *inetd* refuses the connection from the same service invocation five times. This is visible in the system log if *inetd* connection logging and *syslogd* logging for the *daemon* facility are both enabled (see *syslogd*(1M)).

inetd provides several "trivial" services internally by use of routines within itself. The services are **echo**, **discard**, **chargen** (character generator), **daytime** (human readable time), and **time** (machine readable time in the form of the number of seconds since midnight, January 1, 1900). *inetd* provides both tcp- and udp-based servers for each of these services. See *inetd.conf*(4) for instructions on configuring internal servers.

The following options can be used with *inetd*:

-c Reconfigure the Internet daemon; in other words, force the current *inetd* to re-read **/etc/inetd.conf**. This option sends the signal **SIGHUP** to the Internet daemon that is currently running. Any configuration errors that occur during the reconfiguration are logged to the *syslogd daemon* facility.

-l By default, *inetd* starts with connection logging disabled. If no *inetd* is running, the **-l** option causes the *inetd* to start with connection logging enabled. Otherwise the **-l** option causes *inetd* to send the signal **SIGQUIT** to the *inetd* that is already running, which causes it to toggle the state of connection logging.

When connection logging is enabled, the Internet daemon logs attempted connections to services. It also logs connection attempts which fail the security check. This information can be useful when trying to determine if someone is repeatedly trying to access your system from a particular remote system (in other words, trying to break into your system). Successful connection attempts are logged to the *syslogd daemon* facility at the *info* log level. Connection attempts failing the security check are logged at the *notice* log level.

(Requires Optional LAN/X.25 Software)

inetd also logs whether the connection logging has been enabled or disabled at the *info* log level.

-k Kill the current *inetd*. This option sends the signal **SIGTERM** to the Internet daemon that is currently running, causing it to exit gracefully. This option is the preferred method of killing *inetd*.

These options to *inetd* can be used only by the super-user.

DIAGNOSTICS

Diagnostics returned by the Internet daemon before it disconnects from the terminal are:

An inetd is already running
An attempt was made to start an Internet daemon when one is already running. It is incorrect to call the Internet daemon a second time without the **-c**, **-k**, or **-l** option.

There is no inetd running
An attempt was made to reconfigure an Internet daemon when none was running.

Inetd not found
This message occurs if *inetd* is called with **-c** and another Internet daemon is running but cannot be reconfigured. This occurs if the original Internet daemon died without removing its semaphore.

Next step: Do an **inetd -k** to remove the semaphore left by the previous Internet daemon, then restart the daemon.

The following diagnostics are logged to the *syslogd daemon* facility. Unless otherwise indicated, messages are logged at the *error* log level.

/etc/inetd.conf: Unusable configuration file
The Internet daemon is unable to access the configuration file **/etc/inetd.conf**. The error message preceding this one specifies the reason for the failure.

/etc/inetd.conf: line *number*: *error*
There is an error on the specified line in **/etc/inetd.conf**. The line in the configuration file is skipped. This error does not stop the Internet daemon from reading the rest of the file and configuring itself accordingly.

Next step: Fix the line with the error and reconfigure the Internet daemon by executing the command **inetd -c**.

system_call: ...
system call failed. See the corresponding manual entry for a description of *system_call*. The reason for the failure is explained in the error message appended to the system call name.

Cannot configure inetd
None of the services/servers listed in the configuration file could be set up properly, due to configuration file errors.

Too many remote services running
The maximum number of remote services allowed to access the server system simultaneously has been exceeded.

Next step: Consider increasing the number of simultaneous remote services allowed; see *inetd.conf*(4).

file: \ **found before end of line** *line*
file can be either **inetd.conf** or **inetd.sec**. If a backslash is not immediately followed by an end of line, it is ignored and the information up to the end of line is accepted. In this case the next line of the file is not appended to the end of the current line. Unless all the information required is present on a single line, configuration file error messages are also output. This message is logged at the *warning* log level.

service/protocol: **Unknown service**
The call to the library routine *getservbyname* (see *getservent*(3N)) failed. The service is not listed in **/etc/services**.

(Requires Optional LAN/X.25 Software)

Next step: Include that service in **/etc/services** or eliminate the entry for the service in **/etc/inetd.conf**.

service/protocol: **Server failing (looping), service terminated.**
When *inetd* tries to start 40 servers within 60 seconds for a datagram service, it assumes that the server is failing to handle the connection. To avoid entering a potentially infinite loop, *inetd* issues this message, discards the packet requesting the socket connection, and refuses further connections for this service. After 10 minutes, *inetd* tries to reinstate the service, and once again accepts connections for the service.

service/protocol: **socket:** ...
service/protocol: **listen:** ...
service/protocol: **getsockname:** ...
Any one of the three errors above makes the service unusable. For another host to communicate with the server host through this service, the Internet daemon needs to be reconfigured after any of these error messages.

service/protocol: **bind:** ...
If this error occurs, the service is temporarily unusable. After 10 minutes, *inetd* tries again to make the service usable by binding to the Internet socket for the service.

service/protocol: **Access denied to** *remote host* (*address*)
The remote host failed to pass the security test for the indicated service. This information can be useful when trying to determine if someone is repeatedly trying to access your system from a particular remote system (in other words, trying to break into your system). This message is logged at the *warning* log level.

service/protocol: **Connection from** *remote host* (*address*)
When connection logging is enabled, this message indicates a successful connection attempt to the specified service. This message is logged at the *notice* log level.

service/protocol: **Added service, server** *executable*
Keeps track of the services added when reconfiguring the Internet daemon. This message, logged at the *info* log level.

service/protocol: **New** ...
Lists the new user ids, servers or executables used for the service when reconfiguring the Internet daemon. This message is logged at the *info* log level.

service/protocol: **Deleted service**
Keeps track of the services deleted when reconfiguring the Internet daemon. This message is logged at the *info* log level.

Security File (inetd.sec) Errors
The following errors are related to the security file **inetd.sec**:

/usr/adm/inetd.sec: Field contains other characters in addition to * for service
For example, field 2 of the Internet address 10.5*.8.7 is incorrect.

/usr/adm/inetd.sec: Missing low value in range for service
For example, field 2 of the Internet address 10.-5.8.7 is incorrect.

/usr/adm/inetd.sec: Missing high value in range for service
For example, field 2 of the Internet address 10.5-.8.7 is incorrect.

/usr/adm/inetd.sec: High value in range is lower than low value for service
For example, field 2 of the Internet address 10.5-3.8.7 is incorrect.

/usr/adm/inetd.sec: allow/deny field does not have a valid entry for service
The entry in the allow/deny field is not one of the keywords **allow** or **deny**. No security for this service is implemented by *inetd* since the line in the security file is ignored. This message is logged at the *warning* log level.

RPC Related Errors for NFS Users
These errors are specific to RPC-based servers:

(Requires Optional LAN/X.25 Software)

/etc/inetd.conf: line *number*: **Missing program number**
/etc/inetd.conf: line *number*: **Missing version number**
Error on the specified line of **/etc/inetd.conf**. The program or version number for an RPC service is missing. This error does not stop the Internet daemon from reading the rest of the file and configuring itself accordingly. However, the service corresponding to the error message will not be configured correctly.

Next step: Fix the line with the error, then reconfigure the Internet daemon by executing the command **inetd -c .**

/etc/inetd.conf: line *number*: **Invalid program number**
Error on the specified line of **/etc/inetd.conf**. The program number for an RPC service is not a number. This error does not stop the Internet daemon from reading the rest of the file and configuring itself accordingly. However, the service corresponding to the error message will not be correctly configured.

Next step: Fix the line with the error, then reconfigure the Internet daemon by executing the command **inetd -c**.

AUTHOR

inetd was developed by HP and the University of California, Berkeley.
NFS was developed by Sun Microsystems, Inc.

FILES

/etc/inetd.conf	list of Internet server processes.
/usr/adm/inetd.sec	optional security file.

SEE ALSO

umask(1), portmap(1M), syslogd(1M), getservent(3N), inetd.conf(4), inetd.sec(4), protocols(4), services(4), environ(5).

NAME

init, telinit - process control initialization

SYNOPSIS

/etc/init [**0123456SsQq**]

/etc/telinit [**0123456sSQqabc**]

DESCRIPTION

init

init is a general process spawner. Its primary role is to create processes from a script stored in the file **/etc/inittab** (see *inittab*(4)). This file usually has *init* spawn *getty* s on each line that users can log in on. It also controls autonomous processes required by any particular system.

init considers the system to be in a *run level* at any given time. A *run level* can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by *init* for each of these *run levels* is defined in the *inittab* file. *init* can be in one of eight *run level*s, **0-6** and **S** or **s**. The *run level* is changed by having a privileged user run **/etc/init** (which is linked to **/etc/telinit**). This user-spawned *init* sends appropriate signals to the original *init* spawned by the operating system when the system was rebooted, telling it which *run level* to change to.

init is invoked inside the HP-UX system as the last step in the boot procedure. *init* first performs any required machine-dependent initialization, such as setting the system context (see *context*(5)). Next *init* looks for file **/etc/inittab** to see if there is an entry of the type *initdefault* (see *inittab*(4)). If an *initdefault* entry is found, *init* uses the *run level* specified in that entry as the initial *run level* to enter. If this entry is not in *inittab* or *inittab* is not found, *init* requests that the user enter a *run level* from the logical system console, **/dev/syscon**. If an **S** (**s**) is entered, *init* goes into the *single-user* level. This is the only *run level* that does not require the existence of a properly formatted *inittab* file. If **/etc/inittab** does not exist, then by default the only legal *run level* that *init* can enter is the *single-user* level. In the *single-user* level the logical system console terminal **/dev/syscon** is opened for reading and writing, and the command **/bin/su** is invoked immediately. To exit from the *single-user run level*, one of two options can be elected: First, if the shell is terminated (via an end-of-file), *init* reprompts for a new *run level*. Second, the *init* or *telinit* command can signal *init* and force it to change the current system *run level*.

When attempting to boot the system, some processes spawned by *init* may send display messages to the system console (depending on the contents of *inittab*). If messages are expected but do not appear during booting, it may be caused by the logical system console (**/dev/syscon**) being linked to a device that is not the physical system console (**/dev/systty**). If this occurs, *init* can be forced to relink **/dev/syscon** to **/dev/systty** by typing a **delete** on the physical system console.

When *init* prompts for the new *run level*, the operator can enter only one of the digits **0** through **6** or the letter **S** or **s**. If **S** is entered *init* operates as previously described in *single-user* mode with the additional result that **/dev/syscon** is linked to the user's terminal line, thus making it the logical system console. A message is generated on the physical system console, **/dev/systty**, identifying the new logical system console.

When *init* comes up initially, and whenever it switches out of *single-user* state to normal run states, it sets the *ioctl*(2) states of the logical system console, **/dev/syscon**, to those modes saved in the file **/etc/ioctl.syscon**. This file is written by *init* whenever *single-user* mode is entered. If this file does not exist when *init* wants to read it, a warning is printed and default settings are assumed.

If a **0** through **6** is entered *init* enters the corresponding *run level*. Any other input is rejected and a new prompt is issued. If this is the first time *init* has entered a *run level* other than *single-user*, *init* first scans *inittab* for special entries of the type *boot* and *bootwait*. These entries are performed, providing the *run level* entered matches that of the entry before any normal processing of *inittab* takes place. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. The *inittab* file is scanned to find all entries that are to be processed for that *run level*.

Run level **2** is usually defined by the user to contain all of the terminal processes and daemons that are spawned in the multi-user environment.

In a multi-user environment, the *inittab* file is usually set up so that *init* creates a process for each terminal on the system.

For terminal processes, ultimately the shell terminates because of an end-of-file either typed explicitly or generated as the result of hanging up. When *init* receives a child death signal telling it that a process it spawned has died, it records the fact and the reason it died in **/etc/utmp** and **/etc/wtmp** if it exists (see *who*(1)). A history of the processes spawned is kept in **/etc/wtmp** if such a file exists.

To spawn each process in the *inittab* file, *init* reads each entry and, for each entry which should be respawned, it forks a child process. After it has spawned all of the processes specified by the *inittab* file, *init* waits for one of its descendant processes to die, a powerfail signal, or until *init* is signaled by *init* or *telinit* to change the system's *run level*. When one of the above three conditions occurs, *init* re-examines the *inittab* file. New entries can be added to the *inittab* file at any time. However, *init* still waits for one of the above three conditions to occur. To provide for an instantaneous response, the **init Q** or **init q** command can wake *init* to re-examine the *inittab* file.

If *init* receives a *powerfail* signal **SIGPWR** and is not in *single-user* mode, it scans *inittab* for special powerfail entries. These entries are invoked (if the *run levels* permit) before any processing takes place by *init*. In this way *init* can perform various cleanup and recording functions whenever the operating system experiences a power failure. Note, however, that although *init* receives (SIGPWR) immediately after a power failure, *init* cannot handle the signal until it resumes execution. Since execution order is based on scheduling priority, any eligible process with a higher priority executes before *init* can scan *inittab* and perform the specified functions.

When *init* is requested to change *run levels* (via *telinit*), *init* sends the warning signal (**SIGTERM**) to all processes that are undefined in the target *run level*. *init* waits 20 seconds before forcibly terminating these processes via the kill signal (**SIGKILL**). Note that *init* assumes that all these processes (and their descendants) remain in the same process group *init* originally created for them. If any process changes its process group affiliation via either *setpgrp*(2) or *setpgrp2*(2), it will not receive these signals. (Common examples of such a process are *ksh*(1) and *csh*(1)). Such processes need to be terminated separately.

telinit

telinit, which is linked to **/etc/init**, is used to direct the actions of *init*. It takes a one-character argument and signals *init* via the *kill* system call to perform the appropriate action. The following arguments serve as directives to *init*:

0-6 tells *init* to place the system in one of the *run levels* **0** through **6**.

a, b, c tells *init* to process only those **/etc/inittab** file entries having *run level* **a**, **b**, or **c** set.

Q or **q** tells *init* to re-examine the **/etc/inittab** file.

S or **s** tells *init* to enter the single user environment. When this level change is effected, logical system console **/dev/syscon** is changed to the terminal from which the command was executed.

telinit can be invoked only by users with appropriate privileges.

DIAGNOSTICS

If *init* finds that it is continuously respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string, generate an error message on the system console, and refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in the *inittab*.

WARNINGS

init assumes that processes and descendants of processes spawned by *init* remain in the same process group that *init* originally created for them. When changing init states, special care should be taken with processes that change their process group affiliation (*ksh*(1) and *csh*(1) for example).

One particular scenario that often causes confusing behavior can occur when a child *ksh* or *csh* are started by a login shell. When *init* is asked to change to a run level that would cause the original login shell to be killed, the shell's descendant *ksh* or *csh* process does not recieve a hangup signal since it has changed its process group affiliation and is no longer affiliated with the process group of the original shell. *init* cannot kill this *ksh* or *csh* process (or any of its children).

If a *getty*(1M) process is later started on the same tty as this previous shell, the result may be two processes (the getty and job control shell) competing for input on the tty.

To avoid problems such as this, always be sure to manually kill any job control shells that should not be running after changing init states. Also, always be sure that *init* or *telinit* is invoked from the lowest level (login) shell when changing to an init state that may cause your login shell to be killed.

FILES

/etc/clusterconf
/etc/inittab
/etc/ioctl.syscon
/dev/syscon
/dev/systty
/etc/utmp
/etc/wtmp

SEE ALSO

getty(1M), login(1), sh(1), who(1), kill(2), clusterconf(4), inittab(4), utmp(4), context(5).

STANDARDS CONFORMANCE

init: SVID2

NAME

insf - install special files

SYNOPSIS

`/etc/insf`

`/etc/insf` [`-N` *cnode*] [`-d` *driver*] [`-C` *class*] [`-H` *hw_path*] [`-l` *lu*] [`-f`] [`-k`|`-e`] [`-n` *npty*] [`-p` *first_cartridge* `:`*last_cartridge*]

DESCRIPTION

`insf` installs special files in the current directory and assigns logical unit numbers to devices in the system. If no options are specified, special files are created for all new devices in the system. New devices are those devices that have not previously been assigned a logical unit number (i.e., *lu*). A subset of the new devices can be selected with the `-d`, `-C`, and `-H` options.

For each new device in the system, `insf` assigns the next available logical unit number for the device class. The `-l` option can be used to explicitly specify the *lu*. If `insf` is invoked with the `-l` option, the subsetting must be sufficient to identify a unique device in the system. In addition, if the specified *lu* has already been assigned to another device in the device class, `insf` fails unless the `-f` option is specified. The `-f` option forces the *lu* to map to the specified device; any previous mapping of this *lu* to another device is lost.

If the `-k` option is specified, `insf` assigns logical unit numbers without creating special files. Combining the `-l`, `-f`, and `-k` options allows the user to force an existing lu and the corresponding set of special files to map to a different device. This is useful when moving devices around in the system.

If the `-e` option is specified, `insf` reinstalls device files for pseudo-drivers and existing devices; that is, devices which have associated logical unit numbers. This option can be used to restore device files in case one or more have been removed.

The `-N` option specifies that the special files are to be created with the associated *cnode* id; the format of *cnode* is the same as that given in *mknod*(1M). If `-N` is not specified, `insf` uses the cnode id of the machine on which it is executing.

The `-n` option only applies to pty special file installation; that is, drivers `pty0` and `pty1`. If it is given, only *npty* pty special files for each driver will be installed; if it is omitted, 48 pty special files will be installed.

The `-p` option only applies to autochanger special file installation; that is, the driver `autox0orautoch`. If it is given, only special files for those cartridges located in the slots ranging from *first_cartridge* to *last_cartridge* will be installed; if it is omitted, special files for 32 cartridges (1 through 32) will be installed (2048 special files in all).

After assigning logical unit numbers and installing special files, `insf` updates the `/etc/ioconfig` file. If `/etc/ioconfig` is not present, `insf` will create it.

Options

`-C` *class* — Match devices that belong to a given device *class*. Device classes are defined in the `/etc/master` file and can be listed using *lsdev*(1M). The special class `pseudo` includes all pseudo-drivers. Cannot be used with `-d`.

`-H` *hw_path* — Match devices at a given *hw_path*. A hardware path specifies the address of the hardware components leading to a device. It consists of a string of numbers each suffixed by slash (`/`), followed by an arbitrary length string of numbers separated by periods (`.`). Hardware components suffixed by slashes indicate bus converters and may not be necessary on your machine. Hardware components suffixed by `.` indicate the addresses of the remaining hardware components on the path to a device. If the hardware path contains fewer numbers than is necessary to reach a device, special files will be made for all devices connected to the hardware component corresponding to the last number specified.

`-N` *cnode* — Install special files with the specified *cnode* id; the format of *cnode* is the same as that given in *mknod*(1M).

`-d` *driver* — Match devices that are controlled by the specified device *driver*. Device drivers are defined in the `/etc/master` file and can be listed using *lsdev*(1M). Cannot be used with `-C`.

`-e` Reinstall special files for existing devices; that is, devices which have associated logical units. Cannot be used with `-k`.

`-f` Force the *lu* specified in the `-l` option to map to a specific new device; any previous mapping of the *lu* to another device in the device class is lost.

`-k` Assign logical unit numbers without creating special files. Cannot be used with `-e`.

`-l` *lu* Assign the specified *lu* to a single new device in the system.

`-n` *npty* Install *npty* pty special files. This option only applies to `pty0` and `pty1` special file installation.

`-p` *first_cartridge:last_cartridge*
Install special files for those cartridges located in the range *first_cartridge* to *second_cartridge*. This option only applies to `autox0 and autoch` special file installation.

Naming Conventions

The following section shows which special files are created and their permissions for each device driver. File permissions are set by `insf`. Unless otherwise noted, the owner and group ID are set to `bin`.

`autox0/autoch`
Special file names for `autox0 and autoch` use the format: *clu*`d`*surface*`s`*section*. For each logical unit, the following special files are installed:

`ac/`*clu*`d`*surface*`s`*section*
surfaces 1a through 32b, sections 0 through 15 unless otherwise noted by the `-p` option, group `sys`, block entry, `rw- r-- ---`

`rac/`*clu*`d`*surface*`s`*section*
surfaces 1a through 32b, sections 0 through 15 unless otherwise noted by the `-p` option, group `sys`,`character`entry, `rw- r-- ---`

`rac/ioctl`*lu* character entry, `rw- --- ---`

`clone`
The following files are installed:

`strlog`	owner `root`, group `sys`, `rw- rw- rw-`
`sad`	owner `root`, group `sys`, `rw- rw- rw-`
`echo`	owner `root`, group `sys`, `rw- rw- rw-`
`dlpi`	owner `root`, group `sys`, `rw- rw- rw-`

`cn`
The following files are installed:

`syscon`	`rw- -w- -w-`
`systty`	`rw- -w- -w-`
`console`	`rw- -w- -w-`

`devconfig`
`config` owner `root`, group `sys`, `rw- r-- ---`

`diag0`
`diag0 rw- --- ---`

`diaghpib1`
For each logical unit, the following special files are installed:

`diag/hpib/hp28650A/`*lu*
`rw- --- ---`

`disc1`
Special file names for `disc1` use the format: *clu*`d`*unit*`s`*section*. For each logical unit, the following special files are installed:

`dsk/`*clu*`d0s`*section*
sections 0 to 15, group `sys`, block entry, `rw- r-- ---`

rdsk/*clu***d0s***section*
sections 0 to 15, group **sys**, character entry, **rw- r-- ---**
ct/*clu***d***unit***s2**
units 0 and 1, block entry, **rw- rw- rw-**
rct/*clu***d***unit***s2**
units 0 and 1, character entry, **rw- rw- rw-**
diag/dsk/*clu***d***unit*
units 0 and 1, character entry, **rw- --- ---**

disc2
Special file names for **disc2** use the format: ***clu*d*unit*s*section***. For each logical unit, the following special files are installed:

dsk/*clu***d0s***section*
sections 0 to 15, group **sys**, block entry, **rw- r-- ---**
rdsk/*clu***d0s***section*
sections 0 to 15, group **sys**, character entry, **rw- r-- ---**
diag/dsk/*clu***d0**
character entry, **rw- --- ---**

disc3
Special file names for **disc3** use the format: ***clu*d0s*section***. For each logical unit, the following special files are installed:

dsk/*clu***d0s***section*
sections 0 to 15, group **sys**, block entry, **rw- r-- ---**

rdsk/*clu***d0s***section*
sections 0 to 15, group **sys**, character entry, **rw- r-- ---**

display0
For each logical unit, the following special files are installed:

crt*lu* **rw- rw- rw-**
hilkbd*lu* **rw- rw- rw-**
hil_*lu***.***addr* link addresses 1 to 7, **rw- rw- rw-**
tty*ilu* **rw- -w- -w-**
diag/crt*lu* **rw- --- ---**

dlpidrv
dlpi0 owner **root**, group **sys**, **rw- rw- rw-**

dmem
rw- --- ---

gpio0/gpio1
For each logical unit, the following special files are installed:

gpio*lu* **rw- rw- rw-**
diag/gpio*lu* **rw- --- ---**

instr0
For each logical unit, the following special files are installed:

hpib/*lu* **rw- rw- rw-**
hpib/*lu***a***addr*
addrs 0 to 30, **rw- rw- rw-**
diag/hpib/*lu*
rw- --- ---

lan0/lan1
For each logical unit, the following special files are installed:

lan*lu* **rw- rw- rw-**
ether*lu* **rw- rw- rw-**
diag/lan*lu* **rw- --- ---**

lpr0/lpr1/lpr2
For each logical unit, the following special files are installed:

lp*lu*	owner **lp**, **rw- --- ---**
diag/lp*lu*	**rw- --- ---**

meas_drivr

meas_drivr	**rw- rw- rw-**

mirconfig

rdsk/mirconfig	**rw- rw- rw-**

mm
The following special files are installed:

kmem	minor **1**, group **sys**, **rw- r-- ---**
mem	minor **0**, group **sys**, **rw- r-- ---**
null	minor **2**, **rw- rw- rw-**

mux0
For each logical unit, the following special files are installed:

tty*lupport*	ports 0 to 5, direct connect, **rw- -w- -w-**
mux*lu*	**rw- --- ---**
diag/mux*lu*	**rw- --- ---**

mux0_16
For each logical unit, the following special files are installed:

tty*lupport*	ports 0 to 15, direct connect, **rw- -w- -w-**
mux*lu*	**rw- --- ---**
diag/mux*lu*	**rw- --- ---**

mux2
For each logical unit, the following special files are installed:

tty*lupport*	ports 0 to 7, direct connect, **rw- -w- -w-**
mux*lu*	**rw- --- ---**
diag/mux*lu*	**rw- --- ---**

pty0
The following are installed:

pty*index number*
indices **p** through **r**, numbers **0** through **f** (hexadecimal), **rw- rw- rw-**

ptym/pty*index number*
indices **a** through **c** and **e** through **z**, numbers **0** through **f** (hexadecimal), **rw- rw- rw-**

ptym/pty*index number*
indices **a** through **c** and **e** through **z**, numbers **00** through **99**, **rw- rw- rw-**

The special files **pty*** are linked to **ptym/pty***.

pty1
The following are installed:

pty*index number*
indices **p** through **r**, numbers **0** through **f** (hexadecimal), **rw- rw- rw-**

pty/pty*index number*
indices **a** through **c** and **e** through **z**, numbers **0** through **f** (hexadecimal), **rw- rw- rw-**

pty/pty*index number*
indices **a** through **c** and **e** through **z**, numbers **00** through **99**,

rw- rw- rw-

The special files tty* are linked to pty/tty*.

scc1
ttyport ports a and b, direct connect, rw- -w- -w-

sy
tty rw- rw- rw-

tape1/tape2
For each logical unit, the following special files are installed:

rmt/*lu*1 AT&T style, 800 bpi, character entry, rw- rw- rw-
rmt/*lu*m AT&T style, 1600 bpi, character entry, rw- rw- rw-
rmt/*lu*h AT&T style, 6250 bpi, character entry, rw- rw- rw-
rmt/*lu*hc AT&T style, 6250 bpi, compressed, character entry, rw- rw- rw-
rmt/*lu*1n AT&T style, no rewind, 800 bpi, character entry, rw- rw- rw-
rmt/*lu*mn AT&T style, no rewind, 1600 bpi, character entry, rw- rw- rw-
rmt/*lu*hn AT&T style, no rewind, 6250 bpi, character entry, rw- rw- rw-
rmt/*lu*hcn AT&T style, no rewind, compressed, 6250 bpi, character entry, rw- rw- rw-
rmt/*lu*1b Berkeley style, 800 bpi, character entry, rw- rw- rw-
rmt/*lu*mb Berkeley style, 1600 bpi, character entry, rw- rw- rw-
rmt/*lu*hb Berkeley style, 6250 bpi, character entry, rw- rw- rw-
rmt/*lu*hcb Berkeley style, compressed, 6250 bpi, character entry, rw- rw- rw-
rmt/*lu*1nb Berkeley style, no rewind, 800 bpi, character entry, rw- rw- rw-
rmt/*lu*mnb Berkeley style, no rewind, 1600 bpi, character entry, rw- rw- rw-
rmt/*lu*hnb Berkeley style, no rewind, 6250 bpi, character entry, rw- rw- rw-
rmt/*lu*hcnb Berkeley style, compressed, no rewind, 6250 bpi, character entry, rw- rw- rw-
rmt/*lu*qic AT&T style, QIC default format, character entry, rw- rw- rw-
rmt/*lu*qic120 AT&T style, QIC-120 format, character entry, rw- rw- rw-
rmt/*lu*qic150 AT&T style, QIC-150 format, character entry, rw- rw- rw-
rmt/*lu*qic525 AT&T style, QIC-525 format, character entry, rw- rw- rw-
rmt/*lu*qicn AT&T style, no rewind, QIC default format, character entry, rw- rw- rw-
rmt/*lu*qic120n AT&T style, no rewind, QIC-120 format, character entry, rw- rw- rw-
rmt/*lu*qic150n AT&T style, no rewind, QIC-150 format, character entry, rw- rw- rw-
rmt/*lu*qic525n AT&T style, no rewind, QIC-525 format, character entry, rw- rw- rw-
rmt/*lu*qicb Berkeley style, QIC default format, character entry, rw- rw- rw-
rmt/*lu*qic120b Berkeley style, QIC-120 format, character entry, rw- rw- rw-
rmt/*lu*qic150b Berkeley style, QIC-150 format, character entry, rw- rw- rw-
rmt/*lu*qic525b Berkeley style, QIC-525 format, character entry, rw- rw- rw-
rmt/*lu*qicnb Berkeley style, no rewind, QIC default format, character entry, rw- rw- rw-
rmt/*lu*qic120nb Berkeley style, no rewind, QIC-120 format, character entry, rw- rw- rw-
rmt/*lu*qic150nb Berkeley style, no rewind, QIC-150 format, character entry, rw- rw- rw-

`rmt/`*lu*`qic525nb`
Berkeley style, no rewind, QIC-525 format, character entry, `rw- rw- rw-`

`diag/mt/`*lu* character entry, `rw- --- ---`

RETURN VALUE

`insf` returns `0` upon normal completion and `1` if an error occurred. Warnings result in an exit code of `0` because these are not considered catastrophic.

DIAGNOSTICS

Most diagnostic messages from `insf` are self explanatory. Listed below are some messages deserving further clarification. Errors cause `insf` to halt immediately; warnings allow the program to continue.

Errors

Lu *lu* `already assigned to` *driver* `at` *path*
The lu specified with the `-l` option is already assigned to another device in the device class. The `-f` option can be used to force the lu to map to the new device.

`Matched more than one new device`
A logical unit was explicitly specified with the `-l` option and `insf` was unable to match a unique new device. Use some combination of the `-d`, `-C`, and `-H` options to select a unique device in the system.

Warnings

`Device driver` *name* `is not in the kernel`
`Device class` *name* `is not in the kernel`
The indicated device driver or device class is not present in the kernel. A device driver and/or device class can be added to the kernel using *uxgen*(1M).

`No logical units available for device class` *name*
All of the logical units available for the device class are already assigned. Use `rmsf` to remove any unneeded devices from the system (see *rmsf*(1M)).

`Don't know how to handle driver` *name* `- no special files created for` *path*
`insf` does not know how to create special files for the specified device driver. A logical unit number is assigned to the device. Use `mksf` to create special files for the device (see *mksf*(1M)).

EXAMPLES

Install special files and assign logical units for all new devices belonging to the tty device class:

```
insf -C tty
```

Install special files and assign logical unit numbers to the new device added at hardware path 2/4.0.0:

```
insf -H 2/4.0.0
```

WARNINGS

`insf` should only be run in single-user mode. It can change the mode, owner, or group of an existing special file, or unlink and recreate one; special files that are currently open may be left in an indeterminate state.

AUTHOR

`insf` was developed by HP.

FILES

`/dev/config`
`/etc/ioconfig`

SEE ALSO

lsdev(1M), lssf(1M), mksf(1M), mknod(1M), rmsf(1M), uxgen(1M), ioconfig(4).

NAME

install - install commands

SYNOPSIS

/etc/install [**-c** *dira*] [**-f** *dirb*] [**-i**] [**-n** *dirc*] [**-o**] [**-g** *group*] [**-s**] [**-u** *user*] *file* [*dirx ...*]

DESCRIPTION

install is a command most commonly used in "makefiles" (see *make*(1)) to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

install is useful for installing new commands, or new versions of existing commands, in the standard directories (i.e. **/bin**, **/etc**, etc.).

If no options or directories (*dirx ...*) are given, *install* searches a set of default directories (**/bin**, **/usr/bin**, **/etc**, **/lib**, and **/usr/lib**, in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file* (the new version), and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx ...*) are specified after *file*, those directories are searched before the directories specified in the default list.

Options

Options are interpreted as follows:

-c *dira* — Installs a new command (*file*) in the directory specified by *dira*, only if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. Can be used alone or with the **-s** option.

-f *dirb* — Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to **755** and **bin**, respectively. If the file already exists, the mode and owner will be that of the already existing file. Can be used alone or with the **-o** or **-s** options.

-i — Ignores default directory list, searching only through the given directories (*dirx ...*). Can be used alone or with any other options other than **-c** and **-f**.

-n *dirc* — If *file* is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file will be set to **755** and **bin**, respectively. Can be used alone or with any other options other than **-c** and **-f**.

-o — If *file* is found, this option saves the "found" file by copying it to **OLD***file* in the directory in which it was found. This option is useful when installing a normally busy text file such as **/bin/sh** or **/etc/getty**, where the existing file cannot be removed. Can be used alone or with any other options other than **-c**.

-g *group* — Causes *file* to be owned by group *group*. This option is available only to users who have appropriate privileges. Can be used alone or with any other option.

-u *user* — Causes *file* to be owned by user *user*. This option is available only to users who have appropriate privileges. Can be used alone or with any other option.

-s — Suppresses printing of messages other than error messages. Can be used alone or with any other options.

When no directories are specified (*dirx ...*), or when *file* cannot be placed in one of the directories specified, *install* checks for the existence of the file **/etc/syslist**. If **/etc/syslist** exists, it is used to determine the final destination of *file*. If **/etc/syslist** does not exist, the default directory list is further scanned to determine where *file* is to be located.

The file **/etc/syslist** contains a list of absolute pathnames, one per line. The pathname is the "official" destination (for example **/bin/echo**) of the file as it appears on a file system. The file **/etc/syslist** serves as a master list for system command destinations. If there is no entry for *file* in the file **/etc/syslist** the default directory list is further scanned to determine where *file* is to be located.

Cross Generation

The environment variable **ROOT** is used to locate the locations file (in the form **$ROOT/etc/syslist**). This is necessary in cases where cross generation is being done on a production system. Furthermore, each pathname in **$ROOT/etc/syslist** is appended to **$ROOT** (for example, **$ROOT/bin/echo**), and used as the destination for *file*. Also, the default directories are also appended to **$ROOT** so that the default directories are actually **$ROOT/bin**, **$ROOT/usr/bin**, **$ROOT/etc**, **$ROOT/lib**, and **$ROOT/usr/lib**.

The file **/etc/syslist** (**$ROOT/etc/syslist**) does not exist on a distribution tape; it is created and used by local sites.

WARNINGS

install cannot create alias links for a command (for example, *vi*(1) is an alias link for *ex*(1)).

SEE ALSO

make(1), cpset(1M).

NAME

instl_adm - maintain network install message and default parameters

SYNOPSIS

/etc/instl_adm [**-S** *series*] [**-F** *filesystem-kernel*] [**-f** *message-file*] [**-h** *default-host-IP*] [**-s** *default-server-IP*] [**-p** *default-netdist-port*] [**-g** *default-gateway-IP*] [**-m** *default-netmask*]
/etc/instl_adm -d [**-S** *series*] [**-F** *filesystem-kernel*]

DESCRIPTION

instl_adm is executed on a network install boot server system to set or modify the current message and default network parameters used when a client connects to the server to install the HP-UX operating system.

During an HP-UX system installation over a network, the contents of *message-file* are displayed immediately before the user is asked questions about the network to which he or she is connected. The questions that follow the message can also be assigned default answers using **instl_adm**.

Options

instl_adm recognizes the following options:

-S *series* Specifies what series client system the changes apply to. *series* can be any one of **300**, **400**, or **700**. (**300** and **400** are synonymous.) The *series* value specified is independent of the series of server system on which **instl_adm** is executed.

-F *filesystem-kernel*
The informative message and configurable network defaults are stored in *filesystem-kernel*. If the *filesystem-kernel* is not located in the standard place based on the *series* type (see FILES below), use the **-F** option to specify the file path, in which case the **-S** option is not required.

Either the **-S** or the **-F** option must be specified.

-f *message-file*
instl_adm stores the contents of *message-file* for use as the opening message displayed to users doing a network install. If this option is omitted, the current message (if any) is kept. To remove the message completely, simply use an empty file such as **/dev/null** as the *message-file*. Maximum message size is 1024 bytes. To fit on a standard terminal screen, the message should contain not more than 24 lines.

-h *default-host-IP*
Specifies the default Internet host address to be supplied to the install user for his or her system during a network install. Use this option only if a specific Internet address is reserved for install purposes. This should be a temporary Internet address used only during installs. Once the install client has its system installed, it must have its own unique Internet address.

Supplying this default is not recommended if multiple installs are to be done simultaneously because only one active host can occupy a given Internet address at any given time.

default-host-IP cannot be a hostname. It must be a legal Internet address in dot notation such as **15.2.56.1** (see *hosts*(4)).

-s *default-server-IP*
Specifies the default Internet address of a server running **netdistd** from which software should be loaded during a network install. As with *default-host-IP*, *default-server-IP* cannot be a hostname.

-p *default-netdist-port*
Specifies the default port number that the install client should use to connect to the server running **netdistd** (see *netdistd*(1M)).

-g *default-gateway-IP*
Specifies the default Internet address for the gateway system through which the client system can reach the netdist server system (see *route*(1M)). If a gateway is not required, use the keyword **none** for the value of *default-gateway-IP*. As with

default-host-IP, *default-gateway-IP* cannot be a hostname.

`-m` *default-netmask*
Specifies the default netmask for the client system to use to reach the netdist server system. This is necessary if your network uses sub-networks. The netmask is the same as that supplied to the `ifconfig` command (see *ifconfig*(1M)). *default-netmask* can be supplied either in dot-notation (for example, `255.255.248.0`) or as a hexadecimal number with a leading 0x (such as, `0xfffff80`). If sub-networks are not used and the default netmask provided by `ifconfig` is sufficient, use the keyword `none` for the value of *default-netmask*.

WARNING: Specifying any of the options `-h`, `-s`, `-p`, `-g`, or `-m` each of which sets a default network value causes all of the default values to be updated. It is not possible to change one of the values without affecting the others. If any of the options `-h`, `-s`, `-p`, `-g`, or `-m` is specified and any other is not, the default value controlled by the latter is deleted.

`-d` Display current message and default network values.

EXAMPLES

Display the current message and defaults used for serving Series 700 clients:

```
instl_adm -d -S700
```

Change the message to the contents of file `new-message` while leaving the current network defaults in place:

```
instl_adm -S700 -f new-message
```

Replace all the default network values with those given on the command line, except delete the *default-host-IP* value, while leaving the current message in place:

```
instl_adm -S700 -s15.2.40.18 -p2106 -g none -m255.255.248.0
```

FILES

`/usr/lib/uxinstfs.300`
Kernel and file system used by Series 300/400 install clients. The message and defaults for Series 300/400 are kept in this file.

`/usr/lib/uxinstkern.700`
Kernel and file system used by Series 700 install clients. The message and defaults for Series 700 are kept in this file.

SEE ALSO

update(1M), netdistd(1M), ifconfig(1M), route(1M), hosts(4).

NAME

ioinit - initialize I/O system

SYNOPSIS

/etc/ioinit [**-f**] [**-c**] [**-i**]

DESCRIPTION

ioinit initializes kernel I/O system data structures using information from file **/etc/ioconfig**.

If the **-f** option is given, any existing logical unit information in the kernel is erased before being updated from the **/etc/ioconfig** file. Without **-f**, any existing logical unit information in the kernel causes *ioinit* to exit with an error. The **-c** option causes *ioinit* to copy the kernel I/O system data structures into **/etc/ioconfig** instead of initializing the kernel structures from the file. If **/etc/ioconfig** already exists, *ioinit* exits with an error unless **-f** is also specified, in which case the existing **/etc/ioconfig** file is overwritten.

If the **-i** option is given, after successfully initializing the kernel data structures (or copying the structures to **/etc/ioconfig**), *ioinit* runs *insf*(1M) in order to assign logical unit numbers and create special files for all the new devices on the system. Any error encountered before invoking *insf* inhibits the attempt to run it.

Options

ioinit recognizes the following options:

-c Update **/etc/ioconfig** from the kernel instead of the kernel from **/etc/ioconfig**.

-f When used without **-c**, force the information from **/etc/ioconfig** to overwrite any logical unit information already in the kernel. When used with **-c**, force the kernel information to overwrite an existing **/etc/ioconfig** file.

-i Invoke *insf* after updating the kernel data structures or **/etc/ioconfig**.

RETURN VALUE

An exit code of **0** is returned if no errors occurred; **1** is returned if *insf* failed; **2** is returned if *ioinit* encountered an error before invoking *insf*.

Warnings result in an exit code of **0**, since these are not considered catastrophic.

DIAGNOSTICS

Most of the diagnostic messages from *ioinit* are self-explanatory. Listed below are some messages deserving further clarification. Errors cause *ioinit* to halt immediately while warnings allow the program to continue.

If the **-i** option is used, *ioinit* invokes *insf* which may also generate diagnostic messages. Refer to *insf*(1M) for details.

Errors

/etc/ioconfig does not exist
Either the **/etc/ioconfig** file needs to be restored from backup, or the file must be recreated with **ioinit -cf**. If the file is recreated from a kernel without logical unit information in it, that information is lost. In such cases, *insf*(1M) must be run to generate new logical unit numbers, and the new numbers may differ from the original numbers.

Logical unit information is already in the kernel
Existing logical unit information in the kernel is not overwritten unless the **-f** option is specified.

Cannot overwrite existing /etc/ioconfig file
An existing **/etc/ioconfig** file cannot be overwritten unless the **-f** option is specified.

Invalid magic number
The magic number stored in the **/etc/ioconfig** file is incorrect. Either a correct file must restored from backup, or the file must be recreated with **ioinit -cf**. If the file is recreated from a kernel without logical unit information in it, that information is lost. In such cases, *insf*(1M) must be run to generate new logical unit numbers, and the new numbers may differ from the original numbers.

Device driver *name* has conflicting class *name* in /etc/ioconfig
The device class for the given driver as specified in **/etc/ioconfig** does not match the class

specified in the kernel by *uxgen*(1M).

Exec of insf failed
The **-i** option was specified but the attempt to run *insf*(1M) failed for some reason.

Warnings

Device driver *name* **is not in the kernel - device at** *path* **unusable**
Device class *name* **is not in the kernel - device at** *path* **unusable**
Device manager *name* **is not in the kernel - device at** *path* **unusable**
The indicated device driver, device class, or device manager is not present in the kernel and therefore the device at the indicated hardware path is not usable. An *open*(2) of a special file pointing to an unusable device fails. To make the device usable again, the appropriate device driver, device class, and/or device manager must be added to the *uxgen*(1M) input file and a new kernel generated. If the device is no longer needed, *rmsf*(1M) should be used to remove the special files and update **/etc/ioconfig**.

Device driver *name* **is improperly connected - device at** *path* **unusable**
The driver connectivity information specified in **/etc/ioconfig** is different than the connectivity specified in the kernel by *uxgen*(1M) and therefore the device at the indicated hardware path is not usable.

EXAMPLES

For normal system operation, kernel data structures are initialized and special files are made at every boot by the following entry in the **/etc/inittab** file:

```
io::sysinit:/etc/ioinit -i >/dev/console 2>&1
```

AUTHOR

ioinit was developed by HP.

FILES

/etc/ioconfig

SEE ALSO

init(1M), insf(1M), rmsf(1M), uxgen(1M), ioconfig(4).

NAME

ioscan - scan I/O system

SYNOPSIS

Series 800 Only:

/etc/ioscan [**-k**|**-u**] [**-d** *driver*|**-C** *class*] [**-l** *lu*] [**-H** *hw_path*] [**-f** [**-n**]] [*devfile*]
/etc/ioscan -M *module_path* **-H** *hw_path*

Series 300, 400, 700:

/etc/ioscan [**-d** *driver*|**-C** *class*] [**-H** *hw_path*] [**-f**]

DESCRIPTION

Depending on system hardware architecture, **ioscan** scans system hardware, usable I/O system devices, or kernel I/O system data structures as appropriate, and lists the results.

By default, **ioscan** scans the system, and lists all reportable hardware found. The types of hardware reported vary according to what series computer is in the system (see DEPENDENCIES) and can include processors, memory, interface cards and I/O devices. On Series 800 systems, scanning the hardware may cause drivers to be unbound and others bound in their place in order to match actual system hardware. Entities that cannot be scanned are not listed.

Options

ioscan recognizes the following options:

-C *class*	Restrict the output listing to those devices belonging to the specified *class*. Cannot be used with **-d**.
-H *hw_path*	Restrict the scan and output listing to those devices connected at the specified hardware path. When used with **-M**, this option specifies the full hardware path to bind the software modules at.
-M *module_path*	Specifies the module path to bind at the hardware path given by the **-H** option. Must be used with **-H** and cannot be used with any other options.
-d *driver*	Restrict the output listing to those devices controlled by the specified *driver*. Cannot be used with **-C**.
-k	Scan kernel I/O structures instead of the actual hardware. Cannot be used with **-u**.
-l *lu*	Restrict the scan and output listing to the specified logical unit. Must be used with either **-d** or **-C**.
-n	Add device file names to the output listing. Only special files in the **/dev** directory and its subdirectories are listed.
-u	Scan usable I/O system devices instead of the actual hardware. Cannot be used with **-k**.

The **-d** and **-C** options can be used to obtain listings of subsets of the I/O system, but the entire system is still scanned. Specifying **-d** or **-C** along with **-l**, or specifying **-H** or a *devfile* causes **ioscan** to restrict both the scan and the listing to the hardware subset indicated.

On Series 800 systems:

- If **ioscan -k** is used, the kernel I/O system data structures are scanned instead of the actual hardware, and the results are listed. No binding or unbinding of drivers is performed. The **-d**, **-C**, **-l**, and **-H** options restrict listing as described above.
- If **ioscan** is invoked with the **-u** option, usable I/O system devices (those with a driver in the kernel that have a logical unit number assigned) are listed. The **-d**, **-C**, **-l**, and **-H** options restrict the listing as described above.

Output Format

The default output format for **ioscan** lists the class of each hardware module, the hardware path to the module, and the hardware status. If the **-f** option is used, **ioscan** produces a "full" listing, giving the module's class, hardware path, module path, any logical unit, and hardware and software status values. Output fields are as follows:

class Device classes are defined on the series 800 in file **/etc/master** and can be listed using **lsdev** (see *lsdev*(1M)). Examples are **disk**, **printer**, and **tape_drive**.

hw_path A hardware path specifies the address of the hardware components leading to a device. it consists of a string of numbers each suffixed by slash (/), followed by a string of numbers separated by periods (.). Hardware components suffixed by slashes indicate bus converters and may be unnecessary on your machine. Hardware components suffixed by . indicate the addresses of the remaining hardware components on the path to the device.

module_path A module path is a string of arbitrary length consisting of one or more names, separated by periods (.). Each name identifies a driver controlling a hardware component. The string **?** indicates there is no driver available in the system to control that hardware component.

logical_unit The logical unit number associated with the device. It is not listed for those entities that do not have logical units. If no number has been assigned, a dash (-), is listed. This field appears only on Series 800 **ioscan** output listings.

hardware_status
Entity identifier for the hardware component. It is one of the following strings:

ok(0x*idy_value***)**
The hexadecimal value of the entity identifier.

No_Hardware
There is no hardware connected. This cannot occur when doing a hardware scan, but may be encountered when **-k** or **-u** is specified.

Unrecognized_HW
There is hardware present but no entity identifier is available.

Driver_Won't_Probe
No identifier can be obtained because the parent driver does not support probing.

Cannot Probe
No attempt has ever been made to probe this entity. This cannot occur when doing a hardware scan, but may be encountered when **-k** or **-u** is specified.

Probe_Failed
An attempt to probe the hardware failed for some reason.

software_status
The state of the software driver controlling this hardware component. Consists of one of the following strings:

ok The driver is bound.

Unbound
The driver is unbound because another driver has been bound in its place.

Too_Many_Devices
The driver is not bound because the maximum number of bound instances has been reached.

Out_of_Memory
The driver is not bound because the required system resources (such as dynamically allocated memory) could not be obtained.

HW/Driver_Mismatch
The driver is not bound because some other driver should be unbound and this one bound in its place, but the other driver cannot unbind.

No_Driver
There is no driver available in the system that matches this hardware component.

`Cannot_Access`
No `open()` or probe of this hardware component has been done. This cannot occur when doing a hardware scan but may be encountered when `-k` or `-u` is specified.

`Driver_Failure`
The attempt to bind the driver failed for some reason.

`Not_Configured`
The driver does not support dynamic configuration.

In the second Series 800 form shown, `ioscan` forces the specified software module path into the kernel I/O system at the given hardware path, and forces those software modules to be bound. This can be used to make the system recognize a device that could not be recognized automatically; for example, because it has not yet been connected to the system or because diagnostics need to be run on a faulty device.

RETURN VALUE

`ioscan` returns **0** upon normal completion and **1** if an error occurred.

DIAGNOSTICS

Most of the diagnostic messages from `ioscan` are self-explanatory. Listed below are some messages deserving further clarification. Errors cause `ioscan` to halt immediately.

Errors

`Device driver` *name* `is not in the kernel`
`Device class` *name* `is not in the kernel`
The indicated device driver or device class is not present in the kernel. Add the appropriate device driver and/or device class to the *uxgen*(1M) input file and generate a new kernel.

`Invalid module path -` *name1* `cannot connect to` *name2*
The indicated drivers cannot be adjacent in a module path. Driver connectivity is determined by the `/etc/master` file and the drivers "included" in the `uxgen` input file (see *uxgen*(1M)

`No such device in the system`
No device in the system matched the options specified. Use a less specific set of options to list the devices in the system.

EXAMPLES

Scan the system hardware and list all the devices belonging to the disk device class.

```
ioscan -C disk
```

Force-bind the CIO channel adapter driver, HP-IB driver, and tape driver at the hardware path 8.4.1.

```
ioscan -M cio_ca0.hpib0.tape1 -H 8.4.1
```

DEPENDENCIES

Series 300, 400, and 700:

The `-k`, `-u`, `-l`, `-n`, and `-M` options are not supported. Consequently, the second form of `ioscan` given in the synopsis is not valid. No drivers are bound or unbound while scanning the hardware.

The only types of hardware reported are interface cards, disk devices and tape drives. `ioscan` scans the kernel data structures for interface cards, then probes for any disk or tape devices attached.

The `ioscan` output listing differs as follows:

- The *logical unit* field does not exist.
- The field *hardware path* always contains `ok(0x`*idy_value*`).`
- All elements of the hardware path are separated by periods (`.`).
- The field *module path* only lists the controlling driver for a given hardware device, not the full module path.

AUTHOR

`ioscan` was developed by HP.

FILES

`/dev/config`
`/dev/*`

SEE ALSO
lsdev(1M), ioconfig(4).

NAME

isl - initial system loader

DESCRIPTION

isl implements the operating system independent portion of the bootstrap process. It is loaded and executed after self-test and initialization have completed successfully.

The processor contains special purpose memory for maintaining critical configuration related parameters (e.g. Primary Boot, Alternate Boot, and Console Paths). Two forms of memory are supported: Stable Storage and Non-Volatile Memory (NVM).

Typically, when control is transferred to *isl,* an *autoboot* sequence takes place. An *autoboot* sequence allows a complete bootstrap operation to occur with no intervention from an operator. *isl* executes commands from the *autoexecute* file in a script-like fashion. *autoboot* is enabled by a flag in Stable Storage.

autosearch is a mechanism that automatically locates the boot and console devices. For further information, see *pdc*(1M).

During an *autoboot* sequence, *isl* displays its revision and the name of any utility it executes. However, if *autoboot* is disabled, after *isl* displays its revision, it then prompts for input from the console device. Acceptable input is any *isl* command name or the name of any utility available on the system. If a non-fatal error occurs or the executed utility returns, *isl* again prompts for input.

Commands

There are several commands available in *isl.* The following is a list with a short description. Parameters may be entered on the command line following the command name. They must be separated by spaces. *isl* prompts for any necessary parameters that are not entered on the command line.

? **help**	Help - List commands and available utilities
listf **ls**	List available utilities
autoboot	Enable or disable the *autoboot* sequence Parameter - on or off
autosearch	Enable or disable the *autosearch* sequence Parameter - on or off
primpath	Modify the Primary Boot Path Parameter - Primary Boot Path in decimal
altpath	Modify the Alternate Boot Path Parameter - Alternate Boot Path in decimal
conspath	Modify the Console Path Parameter - Console Path in decimal
lsautofl **listautofl**	List contents of the *autoexecute* file
display	Display the Primary Boot, Alternate Boot, and Console Paths
readnvm	Display the contents of one word of NVM in hexadecimal Parameter - NVM address in decimal or standard hexadecimal notation
readss	Display the contents of one word of Stable Storage in hexadecimal Parameter - Stable Storage address in decimal or standard hexadecimal notation

DIAGNOSTICS

isl displays diagnostic information through error messages written on the console and display codes on the LED display.

For the display codes, **CE0***x* are informative only. **CE1***x* and **CE2***x* indicate errors, some of which are fatal and cause the system to halt. Other errors merely cause *isl* to display a message.

Non-fatal errors during an *autoboot* sequence cause the *autoboot* sequence to be aborted and *isl* to prompt for input. After non-fatal errors during an interactive *isl* session, *isl* merely prompts for input.

Fatal errors cause the system to halt. The problem must be corrected and the system **RESET** to recover.

CE00 *isl* is executing.
CE01 *isl* is *autoboot*ing from the *autoexecute* file.
CE02 Cannot find an *autoexecute* file. *autoboot* aborted.
CE03 No console found, *isl* can only *autoboot*.
CE05 Directory of utilities is too big, *isl* reads only 2K bytes.
CE06 *autoexecute* file is inconsistent. *autoboot* aborted.
CE07 Utility file header inconsistent: SOM values invalid.
CE08 *autoexecute* file input string exceeds 2048 characters. *autoboot* aborted.
CE09 *isl* command or utility name exceeds 10 characters.
CE0F *isl* has transferred control to the utility.
CE10 Internal inconsistency: Volume label - **FATAL**.
CE11 Internal inconsistency: Directory - **FATAL**.
CE12 Error reading *autoexecute* file.
CE13 Error reading from console - **FATAL**.
CE14 Error writing to console - **FATAL**.
CE15 Not an *isl* command or utility.
CE16 Utility file header inconsistent: Invalid System ID.
CE17 Error reading utility file header.
CE18 Utility file header inconsistent: Bad magic number.
CE19 Utility would overlay *isl* in memory.
CE1A Utility requires more memory than is configured.
CE1B Error reading utility into memory.
CE1C Incorrect checksum: Reading utility into memory.
CE1D Console needed - **FATAL**.
CE1E Internal inconsistency: Boot device class - **FATAL**.
CE21 Destination memory address of utility is invalid.
CE22 Utility file header inconsistent: *pdc_cache* entry.
CE23 Internal inconsistency: *iodc_entry_init* - **FATAL**.
CE24 Internal inconsistency: *iodc_entry_init* - console - **FATAL**.
CE25 Internal inconsistency: *iodc_entry_init* - boot device - **FATAL**.
CE26 Utility file header inconsistent: Bad aux_id.
CE27 Bad utility file type.

SEE ALSO
boot(1M), hpux_800(1M), pdc(1M).

NAME

killall - kill all active processes

SYNOPSIS

/etc/killall [*signal*]

DESCRIPTION

killall is a procedure used by **/etc/shutdown** to kill all active processes not directly related to the shutdown procedure.

killall is chiefly used to terminate all processes with open files so that the mounted file systems are no longer busy and can be unmounted. **killall** sends the specified *signal* to all user processes in the system, with the following exceptions:

the **init** process;

all processes (including background processes) associated with the terminal from which **killall** was invoked;

any **ps -ef** process, if owned by **root**;

any **sed -e** process, if owned by **root**;

any **shutdown** process;

any **killall** process;

any **/etc/rc** process.

killall obtains its process information from **ps**, and therefore may not be able to perfectly identify which processes to signal (see *ps*(1)).

If no *signal* is specified, a default of **9** (kill) is used.

killall is invoked automatically by **shutdown** The use of **shutdown** is recommended over using **killall** by itself (see *shutdown*(1M)).

FILES

/etc/shutdown

SEE ALSO

fuser(1M), kill(1), ps(1), shutdown(1M), signal(5).

STANDARDS CONFORMANCE

killall: SVID2

NAME

lanconfig - configure network interface parameters

SYNOPSIS

lanconfig *interface* [**ether** | **-ether**] [**ieee** | **-ieee**]

DESCRIPTION

lanconfig is used to configure the network interface protocol. It must be used at boot time to configure each interface present on a machine, and can also be used at a later time to redefine interface protocol configuration. *interface* is a string of the form *name unit*, such as **lan0**.

lanconfig enables or disables protocols as defined by the following parameters:

ieee	Enable IEEE 802.3 protocol over the network interface.
-ieee	Disable IEEE 802.3 protocol over the network interface.
ether	Enable Ethernet protocol over the network interface.
-ether	Disable Ethernet protocol over the network interface.
(none)	If no optional parameters are supplied, **lanconfig** displays the current configuration for *interface*.

Only users with appropriate privileges can modify the configuration of a network interface.

If no protocol encapsulation method is enabled, the lan driver cannot send packets over that interface and any subsequent **send()** fails, returning the error [ENETUNREACH].

DIAGNOSTICS

Messages indicating the specified interface does not exist, the requested parameter value is unknown, or the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

netstat(1), ifconfig(1m), hosts(4), routing(7).

NAME

landiag - local area network diagnostic

SYNOPSIS

`landiag [-e][-t]`

DESCRIPTION

`landiag` is a diagnostic program for testing the Local Area Network (LAN). It allows the user to examine and reset the status of the LAN interface card. LAN interface status includes the LAN interface card self-test completion code, the local network address, and various LAN interface card statistics. Users with appropriate privileges can reset the local LAN interface card, causing it to execute its self-test.

`landiag` reads commands from the standard input, writes prompts and error messages to standard error, and writes status to the standard output. The **Break** key generates an interrupt of a currently executing command if initiated interactively. If commands are read from a file, **Ctrl-|** returns control to the shell.

`landiag` accepts either complete command words or unique abbreviations, and no distinction is made between uppercase and lowercase letters in commands. Multiple commands can be entered on one line if they are separated by spaces, tabs, or commas.

Options

`landiag` recognizes the following command-line options:

`-e` Echo the input commands on the output device.

`-t` Suppress the display of the command menu before each command prompt. This option functions identically to the Test Selection Mode `terse` command. The default for `landiag` is `verbose`.

Test Selection Mode

The available Test Selection Mode commands are:

`lan` Puts `landiag` in LAN Interface Test Mode.

`menu` Displays the Test Selection Mode command menu.

`quit` Terminates the `landiag` program.

`terse` Suppresses display of command menus.

`verbose`
Restores default display of command menus.

LAN Interface Test Mode

The following commands are available:

`clear` Clears the LAN interface card network statistics registers to zero. Requires superuser privileges to execute.

`display`
Displays the local LAN interface card status and statistics registers.

`end` Returns `landiag` to Test Selection Mode.

`menu` Displays the LAN Interface Test Mode command menu.

`name` Prompts for a device file name for the LAN interface card. Default on Series 300/400 systems is `/dev/lan`. Default on Series 700 systems is `/dev/lan0`.

`quit` Terminates the `landiag` program.

`reset` Resets the local LAN interface card which causes it to execute its self-test. Local access to the network is interrupted during execution of `reset`. Requires superuser privileges to execute.

AUTHOR

`landiag` was developed by HP.

FILES

`/usr/bin/landiag`

SEE ALSO
linkloop(1M), lanscan(1M) ping(1M), lan(7).

NAME

lanscan - display LAN device configuration and status

SYNOPSIS

lanscan [*system* [*core*]]

DESCRIPTION

lanscan displays the following information about each LAN device that has software support on the system:

- Series 700/800 Hardware Path or Series 300/400 Select Code.
- Active Station Address (also known as Physical Address).
- Device *lu* (logical unit).
- Hardware State.
- Network Interface "Name Unit" and State.
- Network Management ID.
- Encapsulation Methods configured for the Network Interface.
- Major Number of the lan device file. A **--** implies that a major number does not apply to this LAN device.

The arguments *system* and *core* allow substitution for the default values **/hp-ux** and **/dev/kmem**.

WARNINGS

The Device *lu* does not necessarily match the Network Interface Unit on Series 800 computers.

lanscan does not display information about LAN devices that do not have software support such as LAN interface cards that fail to bind properly at boot-up time.

AUTHOR

lanscan was developed by HP.

SEE ALSO

ifconfig(1M), lanconfig(1M), Series 800 only: ioscan(1M).

NAME

last, lastb - indicate last logins of users and ttys

SYNOPSIS

/etc/last [**-c**] [**-R**] [*-count*] [*name ...*] [*tty ...*]
/etc/lastb [**-c**] [**-R**] [*-count*] [*name ...*] [*tty ...*]

DESCRIPTION

last searches backwards through file **/etc/wtmp** (which contains a record of all logins and logouts) for information about a user, a tty, or any group of users and ttys. Arguments specify names of users or ttys of interest. Names of ttys can be given fully or abbreviated. For example, **last 0** is the same as **last tty0**. If multiple arguments are given, the information that applies to any of the arguments is printed. For example, **last root console** lists all of root's sessions as well as all sessions on the console terminal. *last* prints the sessions of the specified users and ttys, most recent first, indicating when the session began, the duration of the session, and the tty on which the session took place. If the session is still continuing or was cut short by a reboot, *last* so indicates.

The pseudo-user **reboot** logs each time the system reboots. Thus **last reboot** is a useful command for evaluating the relative time between system reboots.

last recognizes the following options and arguments:

(none) If no arguments are specified, *last* prints a record of all logins and logouts in reverse order, most recent first.

-c Displays login information for all cnodes of an HP Cluster (see *glossary*(9)).

-count Limits the report to *count* lines. When used in conjunction with the **-c** option, the limit is applied independently for each cnode in the cluster.

-R When used with *last* and *lastb*, **-R** displays the user's host name as it is stored in files **/etc/wtmp** and **/etc/btmp** respectively. The host name is displayed between the tty name and the user's login time.

If *last* is interrupted, it indicates how far the search has progressed in *wtmp*. If interrupted by a quit signal (generated by a Ctrl-\), *last* indicates how far the search has progressed, then continues the search.

lastb searches backwards through the database file **/etc/btmp** to display bad login information. Access to **/etc/btmp** should be restricted to users with appropriate privileges (owned by and readable only by `root`) because it may contain password information.

AUTHOR

last was developed by the University of California, Berkeley and HP.

FILES

/etc/btmp bad login data base
/etc/wtmp login data base

SEE ALSO

login(1), utmp(4).

NAME

lb_admin - Location Broker administrative tool

SYNOPSIS

`/etc/ncs/lb_admin` [`-nq`] [*-version*]

DESCRIPTION

`lb_admin` administers the registrations of NCS-based servers in Global Location Broker (GLB) or Local Location Broker (LLB) databases. A server registers Universal Unique Identifiers (UUIDs) specifying an object, a type, and an interface, along with a socket address specifying its location. A client can locate servers by issuing lookup requests to GLBs and LLBs. `lb_admin` can be used to look up information, add new entries, and delete existing entries in a specified database.

`lb_admin` is useful for inspecting the contents of Location Broker databases and for correcting database errors. For example, if a server terminates abnormally without unregistering itself, use `lb_admin` to manually remove its entry from the GLB database.

In accepting input or displaying output, `lb_admin` uses either character strings or descriptive textual names to identify objects, types, and interfaces. A character string directly represents the data in a UUID in the format *xxxxxxxxxxxx.xx.xx.xx.xx.xx.xx.xx.xx*, where each *x* is a hexadecimal digit. Descriptive textual names are associated with UUIDs in the `uuidname.txt` file.

`lb_admin` examines or modifies only one database at a time, refered to as the "current database". The `use_broker` command selects the type of Location Broker database, GLB or LLB. The `set_broker` command selects the host whose GLB or LLB database is to be accessed. Of course, if you modify one replica of a replicated GLB database, the modifications is propagated to the other replicas of that database.

Options

`lb_admin` recognizes the following options:

`-nq` Do not query for verification of wildcard expansions in `unregister` operations.

-version Display the version of NCK that this `lb_admin` belongs to, but do not start the tool.

Commands

In `lookup`, `register`, and `unregister` commands, the *object*, *type*, and *interface* arguments can be either character strings representing UUIDs or textual names corresponding to UUIDs, as described earlier.

`a[dd]` Synonym for `register`.

`c[lean]` Find and delete obsolete entries in the current database.

With this command, `lb_admin` attempts to contact each server registered in the database. If the server responds, the entry for its registration is left intact in the database. If the server does not respond, `lb_admin` tries to look up its registration in the LLB database at the host where the server is located, displays the result of this lookup, and asks whether to delete the entry. If a server responds, but its UUIDs do not match the entry in the database, `lb_admin` displays this result and asks whether to delete the entry.

There are two situations in which it is likely that a database entry should be deleted:

- The server does not respond, `lb_admin` succeeds in contacting the LLB at the host where the server is located, and the server is not registered with that LLB. The server is probably no longer running.
- A server responds, but its UUIDs do not match the entry in the database. The server that responded is not the one that registered the entry.

Entries that meet either of these conditions are probably safe to delete and are considered eligible for "automatic deletion" (described in the next paragraph). In other situations, it is best not to delete the entry unless you can verify directly that the server is not running (for example, by listing the processes running on its host).

When `lb_admin` asks whether you want to delete an entry, you have four ways to respond. `y[es]` deletes the entry. `n[o]` leaves the entry intact in the database. After a `yes` or a `no`, `lb_admin` proceeds to check the next entry in the current database. `g [ o ]` invokes automatic deletion in which all eligible entries are deleted and all ineligible entries are left intact without your being queried,

until all entries have been checked. **q[uit]** terminates the **clean** operation.

d[elete]
Synonym for **unregister**.

h[elp] [*command*] or **?** [*command*]
Display a description of the specified *command* or, if none is specified, list all of the **lb_admin** commands.

l[ookup] *object type interface*
Look up and display all entries with matching *object*, *type*, and *interface* fields in the current database. You can use asterisks as wildcards for any of the arguments. If all the arguments are wildcards, **lookup** displays the entire database.

q[uit]
Exit the *lb_admin* session.

r[egister] *object type interface location annotation* [*flag*]
Add the specified entry to the current database. Use an asterisk to represent a nil UUID in the *object*, *type*, and *interface* fields.

The *location* is a string in the format *family*:*host*[*port*], where *family* is an address family, *host* is a host name, and *port* is a port number. Possible values for *family* include **ip** and **dds**. Use a leading **#** to indicate that a host name is in the standard numeric form. For example, **ip:vienna[1756]**, **ip:#192.5.5.5[1791]**, **dds://salzburg[515]**, and **dds:#575d.542e[452]** are all acceptable *location* specifiers. (All HP-UX hosts have **ip** as the *family*.)

The *annotation* is a string of up to 64 characters annotating the entry. Use double quotation marks to delimit a string that contains a space or contains no characters. To embed a double quotation mark in the string, precede it with a backslash.

The *flag* is either **local** (the default) or **global**, indicating whether the entry should be marked for local registration only or for registration in both the LLB and GLB databases. The *flag* is a field that is stored with the entry; it does not affect where the entry is registered. The **set_broker** and **use_broker** commands select the particular LLB or GLB database for registration.

s[et_broker] [*broker_switch*] *host*
Set the host for the current LLB or GLB. If you specify **global** as the *broker_switch*, **set_broker** sets the current GLB; otherwise, it sets the current LLB. The *host* is a string in the format *family*:*host*, where *family* is an address family and *host* is a host name. Possible values for *family* include **ip** and **dds**. Use a leading **#** to indicate that a host name is in the standard numeric form. For example, **ip:prague**, **ip:#192.5.5.5**, **dds://linz**, and **dds:#499d.590f** are all acceptable *host* specifiers. (All HP-UX hosts have **ip** as the *family*.)

Issue **use_broker**, not this command, to determine whether subsequent operations will access the LLB or the GLB.

set_t[imeout] [**short** | **long**]
Set the timeout period used by **lb_admin** for all of its operations. With an argument of **short** or **long**, **set_timeout** sets the timeout accordingly. With no argument, it displays the current timeout value.

u[nregister] *object type interface location*
Delete the specified entry from the current database.

location is a string in the format *family*:*host*[*port*], where *family* is an address family, *host* is a host name, and *port* is a port number. Possible values for *family* include **ip** and **dds**. Use a leading **#** to indicate that a host name is in the standard numeric form. For example, **ip:vienna[1756]**, **ip:#192.5.5.5[1791]**, **dds://salzburg[515]**, and **dds:#575d.542e[452]** are all acceptable *location* specifiers. (All HP-UX hosts have **ip** as the *family*.)

Use an asterisk as a wildcard in the *object*, *type*, and *interface* fields to match any value for the field. Unless **lb_admin -nq** was used to suppress queries, **unregister** asks whether to delete each matching entry. **y[es]** deletes the entry. **n[o]** leaves the entry in the database. **g[o]** deletes all remaining database entries that match, without further interaction. **q[uit]** terminates the **unregister** operation without deleting any more entries.

us[e_broker] [*broker_switch*]
Select the type of database that subsequent operations will access, GLB or LLB. The *broker_switch* is either **global** or **local**. If *broker_switch* is not specified, **use_broker** tells whether the current database is global or local.

Use **set_broker** to select the host whose GLB or LLB is to be accessed.

SEE ALSO
drm_admin(1M), glbd(1M), llbd(1M), uuidname.txt(4).

NAME

lb_test - test the Location Broker

SYNOPSIS

`/etc/ncs/lb_test` [`-unreg`] [`-num` *entries*] [`-time` *seconds*]

DESCRIPTION

lb_test tests the functionality of the Global Location Broker (GLB) and Local Location Broker (LLB), parts of the Network Computing System (NCS).

`lb_test` repeatedly registers and looks up entries with the GLB and with the local LLB. It fails if a GLB or LLB does not exist, is an incorrect version, is inaccessible, or is functioning improperly. The host that runs `lb_test` must also run the Local Location Broker daemon (`llbd`) and must be able to reach a Global Location Broker daemon (`glbd` or `nrglbd`).

A Location Broker entry contains Universal Unique Identifiers (UUIDs) for an object, the type of the object, and an interface to the object, together with the location of a server exporting the interface. To avoid flooding the GLB and LLB databases, `lb_test` always uses the same three UUIDs for all registrations.

`lb_test` loops, making 256 passes. Each iteration of the loop attempts to register a set of entries and look them up. `lb_test` attempts to unregister the entries when it exits.

Command-line options specify the number of entries that `lb_test` registers at each iteration and the amount of time that it waits between iterations. If options are not specified, `lb_test` uses default values.

If `lb_test` fails, try running `lb_admin` to access a Global Location Broker; if `lb_admin` also fails, there is probably a problem in the GLB configuration.

Options

`-num` *entries*
: Register the specified number of entries. The default for *entries* is 5.

`-time` *seconds*
: Sleep for the specified number of seconds between iterations. The default is no waiting.

`-unreg`
: Unregister all entries of `lb_test` type UUID. After unregistering the entries, `lb_test` terminates.

Sample Output

This section shows the output that `lb_test` produces if the Location Broker is operating correctly. Most of the output has been omitted as indicated by the ellipses (`...`).

If the hosts and/or the network used in the test are heavily loaded, additional informational messages may be printed. These message are printed when the Remote Procedure Call (RPC) runtime library is recovering from slow transmissions or lost packets.

```
$ lb_test
Handling 5 registrations with no time delay
Starting test loop...
    Pass 0
    Pass 1
    Pass 2
    Pass 3
        .
        .
        .
    Pass 252
    Pass 253
    Pass 254
    Pass 255
Unregistering...
$
```

SEE ALSO

glbd(1M), nrglbd(1M), llbd(1M), perf(1M).

NAME

link, unlink - exercise link and unlink system calls

SYNOPSIS

`/etc/link` *file1 file2*
`/etc/unlink` *file*

DESCRIPTION

`link` and `unlink` perform their respective system calls on their arguments, abandoning most error checking. These commands can be executed only by users who have appropriate privileges.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

`link` and `unlink` return the following values:

0 Operation successful.
1 Input syntax error.
2 `link` or `unlink` call failed.

WARNINGS

If a directory that is *not* empty (contains files other than `.` and `..`), is unlinked, the files become orphans unless they are linked by some other directory.

SEE ALSO

rm(1), link(2), unlink(2).

STANDARDS CONFORMANCE

`link`: SVID2
`unlink`: SVID2

NAME

linkloop - verify LAN connectivity with link-level loopback

SYNOPSIS

linkloop [**-n** *count*] [**-f** *devfile*] [**-t** *timeout*] [**-s** *size*] [**-r** *rif*] [**-v**] *linkaddr1* [*linkaddr2* ...]

DESCRIPTION

linkloop uses IEEE 802.2 link-level test frames to check connectivity within a local area network (LAN). This program differs from the remote loopback capability of **rlb** (see *rlb*(1M)) in that it tests only link-level connectivity; not transport-level connectivity.

The required parameter is the hardware station address (*linkaddr1*, *linkaddr2*, ...) of a remote node. **linkloop** tests the connectivity of the local node and the remote node specified by the hardware station address. The hardware station address of a remote node can be found by executing **lanscan** on the remote node. This hardware station address is usually represented as a hexadecimal string prefixed with **0x**, but can also be represented as a octal string prefixed with **0** or as a decimal string. The hardware station address must not be a multicast or broadcast address.

Options

linkloop recognizes the following options:

-n *count* Sets the number of frames to transmit. If *count* is **0**, **linkloop** transfers frames indefinitely until an interrupt signal (defined by the user shell) is received. The default value for *count* is **1**.

-t *timeout* Sets the amount of time (in seconds) to wait for a reply from the remote node before aborting. If *timeout* is **0**, **linkloop** waits indefinitely for a reply. The default value for *timeout* is 2 seconds.

-s *size* Sets the size of the data message to send. The maximum data size is dependent on the type of LAN link being used. The default value is the maximum data byte count that can be used for the particular link.

-v Sets the verbose option. In addition to the regular summary of test results, this option displays more extensive error information. If there are header or length errors, appropriate messages are displayed. All verbose output is preceded by the number of replies accepted before an error occurred.

-f *devfile* Specifies which device file to use. The device file must be an LAN device file. If no device file is entered, **linkloop** uses **/dev/lan** (Series 300/400) as the default. If **/dev/lan** is not present or is the wrong type of device file, **linkloop** uses **/dev/ieee** (Series 300/400) as the default. If no device file is entered, **linkloop** uses **/dev/lan0** (Series 700) as the default. If **/dev/lan0** is not present or is the wrong type of device file, **linkloop** uses **/dev/ieee0** (Series 700) as the default.

-r *rif* The *rif* is the *routing information* field used for token-ring networks. This information allows a user to specify the particular bridge route over which the token ring packet should be delivered. The *rif* value is given as an even number of hexadecimal bytes separated by colons, up to a maximum of 16 bytes.

Connectivity Test Results

linkloop aborts upon receipt of an interrupt signal. If aborted, the current results are printed.

linkloop prints the result of the link-level connectivity test. If the test fails, it prints a summary of the test and indicates the type of error. The possible messages are:

address has bad format
Incorrect hardware station address was entered on the command line.

address is not individual
Station address entered on the command line is either a multicast or broadcast address.

frames sent
Total number of frames sent.

frames received correctly
Total number of frames received without errors.

`frames with length error`
Received frame length does not match transmitted frame length. If the verbose option is set, the length received is printed.

`frames with data error`
Received frame does not match transmitted frame.

`frames with header error`
Number of frames received containing unexpected frame header information. Either the source address does not match the remote address, the destination address does not match the local address, or the control field is not the TEST **frame control field**. These frames are ignored. `linkloop` continues to try to receive the reply frame until the `read` operation times out.

`reads that timed out`
Count of how many `read` operations timed out before the reply was received.

DIAGNOSTICS

`illegal count parameter`
count is not a non-negative integer, or the number specified is too large for the local computer.

`illegal timeout parameter`
timeout is not a non-negative integer, or the number specified multiplied by 1000 is too large for the local computer.

`illegal size parameter`
Size specified is not in the range from 0 to the maximum link data size. Remember that the maximum link data size may vary in value between different LAN connection types.

`unable to use device file`
Specified device file does not exist or is of the wrong type, or `linkloop` is already being executed by another user.

`unable to use default device file`
Default device file does not exist or is of the wrong type, or `linkloop` is already being executed by another user.

`invalid rif parameter`
Values in the rif parameter were invalid.

`rif parameter too long`
The number of bytes in the rif parameter exceeded the maximum allowed (16).

`rif parameter length must be even`
The number of bytes in the rif parameter was odd. The number of bytes must be even.

AUTHOR

`linkloop` was developed by HP.

FILES

`/dev/lan`	first default device file for Series 300/400
`/dev/ieee`	second default device file for Series 300/400
`/dev/lan0`	first default device file for Series 700
`/dev/ieee0`	second default device file for Series 700

SEE ALSO

lanscan(1M), lan(7).

NAME

llbd - Local Location Broker daemon

SYNOPSIS

/etc/ncs/llbd [*-version*]

DESCRIPTION

The Local Location Broker daemon (*llbd*) is part of the Network Computing System (NCS). It manages the Local Location Broker (LLB) database, which stores information about NCS-based server programs running on the local host.

A host must run *llbd* if it is to support the Location Broker forwarding function or to allow remote access (for example, by the *lb_admin* tool) to the LLB database. In general, any host that runs NCS-based servers should run an *llbd*, and *llbd* should be running before any such servers are started. Additionally, any network or internet supporting NCS activity should have at least one host running a Global Location Broker daemon (*glbd*).

(On MS-DOS systems, which can run only one process at a time, *llbd* functionality is automatically incorporated into every NCS-based server.)

Running llbd on HP-UX Systems

On HP-UX systems, **llbd** is typically started in **/etc/netncsrc**; if the **START_LLBD** variable in **netncsrc** is set to 1, an *llbd* will be started. To start *llbd* by hand, you must be root.

OPTIONS

-listen *family_list*

This option restricts the address families on which an LLB listens. Use it only if you are creating a special configuration where access to an LLB is restricted to a subset of hosts in the network or internet.

The *family_list* is a list of the address families on which the LLB will listen. Names in this list are separated by spaces. Possible family names include **dds** and **ip**.

If *llbd* is started without the **-listen** option, the LLB will listen on all address families that are supported both by NCS and by the local host. On Domain/OS systems, this set of families always includes **dds** and may also include **ip**. On most other systems, including HP-UX systems, **ip** is currently the only family.

-version

Display the version of NCK that this *llbd* belongs to, but do not start the daemon.

SEE ALSO

glbd(1M), lb_admin(1M).

NAME

localedef - generate and display locale.inf file

SYNOPSIS

`localedef` [`-c`][`-f` *charmap_file*][`-i` *input_file*][*locale_name*]

`localedef -d`[`-o`*format*] *locale_name*

`localedef -n`[`-i` *input_file*]

DESCRIPTION

`localedef` sets up the language environment for the named locale. `localedef` reads a **localedef script** (see *localedef*(4) for a detailed description) from standard input (default) or from *input_file*, creates a file called `locale.inf`, and installs this file in the appropriate directory.

Options

`localedef` recognizes the following options:

`-c` Create permanent output even if warning messages have been generated.

`-d` (Dump) Display contents of `locale.inf` file representing named locale. The contents are written to standard output in a form suitable for input to `localedef` so that the locale can be modified and recreated. The `locale.inf` file is searched for at pathname `/usr/lib/nls/`*locale_name*. If this option is specified, the `-f`, `-i`, and `-n` options are meaningless.

`-f` *charmap_file*
Use *charmap_file* to interpret symbolic names (of the form <*name*>) in the localedef script. This option must be used if symbolic names are used in the localedef script. See *charmap(4)* for a description of the format of a *charmap_file*.

`-i` *input_file* Use *input_file* as the source of the localedef script instead of standard input.

`-n` *input_file* (noinstall) Create the `locale.inf` file in the current directory, using *input_file* as the source localedef script.

`-o` *format* Specify the format of character output with the `-d` option. This option has no effect except when used with `-d`. The arguments *c, d, o* and *x* options cause `localedef` to output each non-printable character code in the "character constant", "decimal constant", "octal constant", and "hexadecimal constant" form, respectively. Without the `-o` option, `localedef` displays each printable character code in the "character constant" form and non-printable character code in the "hexadecimal constant" form. The character constants have the form:

Type	Form	
c	*c*	where c is a character
d	`\d`*c*[*c*[*c*]]	where c is a decimal digit
o	`\`*c*[*c*[*c*]]	where c is an octal digit
x	`\x`*cc*	where c is a hexadecimal digit

locale_name
This argument is not required, and is ignored, if the *input_file* contains the `langname` keyword (see *localedef*(4)). Otherwise, *locale_name* identifies the name of the language following the naming convention of the **LANG** environment variable (see *environ*(5)):

language[*_territory*][*.codeset*]

This is a brief description of the components that make up a locale. For a complete description of the form and syntax of a localedef script, see *localedef*(4). For a complete description of the form and effects of a charmap file, see *charmap*(4).

Seven categories of data in the `locale.inf` file are recognized by *setlocale*(3C), and make up a language definition:

`LC_COLLATE` Information in this category affects behavior of regular-expressions and NLS string-collation functions.

LC_CTYPE	Information in this category affects behavior of character classification and conversion functions.
LC_MONETARY	Information in this category affects behavior of functions that handle monetary values.
LC_NUMERIC	Information in this category affects handling of the radix character in formatted-input/output and string-conversion functions.
LC_TIME	Information in this category affects behavior of time-conversion functions.
LC_MESSAGES	This category contains information affecting interpretation of yes/no responses.
LC_ALL	This category contains language-specific information that does not belong to any of the above categories.

A **localedef script** also consists of seven categories. The beginning of each category is identified by a **category tag** having the form LC_*category* where *category* is one of the following: CTYPE, COLLATE, MONETARY, NUMERIC, TIME, MESSAGES, or ALL. The end of each category is identified by a tag consisting of the word END followed by a space and the category identifier; for example, END LC_ALL. Categories can appear in any order in the ocaledef script except that, for POSIX.2 conformance, LC_ALL should be last. All category specifications are optional. If a category is not specified, `setlocale()` sets up the default "C" locale for that category (see *setlocale*(3C) and *lang*(5)).

Each category is composed of one or more statements. Each statement begins with a keyword followed by one or more expressions. An expression is a set of well-formed metacharacters, strings, and constants. `localedef` also recognizes comments and separators.

More than one definition can be specified for each category. If a category contains more than one definition, each additional definition must be named via the `modifier` keyword described in *localedef*(4). The first set of specifications is the default definition which might or might not have a modifier name.

Any category can be specified by the keyword `copy` followed by the name of a valid locale. This causes the information for the category to be identical to that in the named locale. Note that the `copy` keyword, if used for a category, must be the first and only keyword following the category tag.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the locale to use when neither LC_ALL or the other category variables specify a locale.

LC_ALL determines locale to be used. It overrides any values specified by LANG or any other LC_* variables.

LC_CTYPE determines the printable characters when the `-d` option is specified. It also affects interpretation of characters in arguments as single- or multi-byte. LC_CTYPE has no effect on the processing of localedef, which behaves as if LC_CTYPE were set to the C locale.

If LC_CTYPE is not specified in the environment or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LC_CTYPE.

LC_MESSAGES determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

`localedef returns the following values:`

0	No errors occurred and the locale was successfully created.
1	Warnings occurred and the locale was successfully created.
2	The locale specification exceeded implementation limits or the coded character set used is not supported.
>3	Warnings or errors occurred, and no output was generated.

AUTHOR

`localedef` was developed by HP.

FILES

`/usr/lib/nls/config`
`/usr/lib/nls/`*language*`[/`*territory*`][/`*codeset*`]/locale.inf`

SEE ALSO

locale(1), localedef(4), charmap(4), setlocale(3C), environ(5).

STANDARDS CONFORMANCE

localedef: POSIX.2, XPG4

NAME

lockd - network lock daemon

SYNOPSIS

/usr/etc/rpc.lockd [**-l** *log_file*] [**-t** *timeout*] [**-g** *graceperiod*]

DESCRIPTION

lockd is an RPC server that processes NFS file locking requests from the local kernel or from another remote lock daemon. **lockd** forwards lock requests for remote data to the server site's lock daemon through the RPC/XDR package (see *rpc*(3C)). **lockd** then requests the status monitor daemon, **statd** for monitor service (see *statd*(1M)). The reply to the lock request is not sent to the kernel until the status daemon and the server site's lock daemon have replied.

If either the status monitor or server site's lock daemon is unavailable, the reply to a lock request for remote data is delayed until all daemons become available.

When a server recovers, it waits for a grace period for all NFS client-site **lockd**s to submit reclaim requests. Client-site *lockd*s are notified by the **statd** of the server recovery, and promptly resubmit previously granted lock requests. If a **lockd** fails to secure a previously granted lock at the server site, the **lockd** sends a **SIGLOST** to the process holding that lock.

Options

lockd recognizes the following options and command-line arguments:

-l *log_file* — Log any errors to the named log file *log_file*. Errors are not logged if the **-l** option is not specified.

Information logged to the file includes date and time of the error, host name, process ID and name of the function generating the error, and the error message.

-t *timeout* — **lockd** uses *timeout* (seconds) as the interval instead of the default value (10 seconds) to retransmit a lock request to the remote server. Note that changing this value also changes the value for grace period duration.

-g *graceperiod* — **lockd** uses [1+(*graceperiod*/*timeout*)]×*timeout* (seconds) as the grace period duration instead of the default value (5×*timeout* seconds). If both **-t** and **-g** are specified, the **-t** should appear first since the grace period duration is dependent on the value of timeout.

AUTHOR

lockd was developed by Sun Microsystems, Inc., and HP.

SEE ALSO

fcntl(2), lockf(2), signal(2), statd(1M).

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages

NAME

lpadmin - configure the LP spooling system

SYNOPSIS

/usr/lib/lpadmin -p*printer* [*options*]
/usr/lib/lpadmin -x*dest*
/usr/lib/lpadmin -d[*dest*]

DESCRIPTION

lpadmin configures LP spooling systems to describe printers, classes and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs, and to change the system default destination. *lpadmin* cannot be used when the LP scheduler, *lpsched*(1M), is running, except where noted below.

Exactly one of the **-p**, **-x** or **-d** options must be present for every legal invocation of *lpadmin*.

-p*printer* Names a *printer* to which all of the *options* below refer. If *printer* does not exist, it will be created.

-x*dest* Removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, the class is deleted, too. No other *options* are allowed with **-x**.

-d[*dest*] Makes existing destination *dest* the new system default destination. If *dest* is not supplied, there is no system default destination. This option can be used when *lpsched*(1M) is running. No other *options* are allowed with **-d**.

The following *options* are only useful with **-p** and can appear in any order. For ease of discussion, the printer is referred to below as printer *P*.

-a*cluster_client* Indicates that the printer *P* is attached to the specified *cluster client*. If omitted in a clustered environment, it assumes that printer *P* is attached to the cluster server.

-c*class* Inserts printer *P* into the specified *class*. *class* is created if it does not already exist.

-e*printer* Copies an existing *printer*'s interface program to be the new interface program for printer *P*.

-g*priority* Sets the default priority for printer *P* associated with *lp*(1). If omitted, the default priority is set to 0.

-h Indicates that the device associated with printer *P* is hardwired. This *option* is assumed when creating a new printer unless the **-l** option is specified.

-i*interface* Establishes a new interface program for printer *P*. *interface* is the pathname of the new program.

-l Indicates that the device associated with printer *P* is a login terminal. The LP scheduler (see *lpsched*(1M)) disables all login terminals automatically each time it is started. Before re-enabling printer *P*, its current *device* should be established using *lpadmin*.

-m*model* Selects a model interface program for printer *P*. *model* is one of the model interface names supplied with the LP software (see Models below).

-r*class* Removes printer *P* from the specified *class*. If printer *P* is the last member of the *class*, the *class* is removed.

-v*device* Associates a new *device* with printer *P*. *device* is the pathname of a file that is writable by the LP administrator *lp*. Note that there is nothing to stop an administrator from associating the same *device* with more than one *printer*. If only the **-p** and **-v** *options* are supplied, *lpadmin* can be used while the scheduler is running. If the **-a** option is supplied, *device* should be the pathname of a device file that is on the cluster client.

The following *options* are only useful with **-p** and can appear in any order. They are provided with systems that provide remote spooling.

-ob3 Uses three-digit request numbers associated with the printer directory. This is for contact with BSD systems. The default is to not use three-digit request numbers.

-oci*remcancel* Specifies that the local command *remcancel* is used to cancel requests to remote printers. To ensure that the correct command is used, specify the full path name.

-ocm*remcancel* Specifies that the local model *remcancel* is used to cancel requests to remote printers.

-orm*machine* The name of the remote machine is *machine*.

-orp*printer* The name of the printer to use on the remote machine is *printer*.

-orc Restricts users to canceling only their own requests. Default is to not restrict the cancel command.

-osi*remstatus* Specifies that the command *remstatus* is used to obtain the status of requests to remote printers. To ensure that the correct command is used, specify the full path name.

-osm*remstatus* Specifies that the model *remstatus* is used to obtain the status of requests to remote printers.

Restrictions

When creating a new printer, the **-v** option and one of the **-e**, **-i** , or **-m** options must be specified. Only one of the **-e**, **-i** or **-m** options can be specified. The **-h** and **-l** key letters are mutually exclusive. The **-a** and **-l** key letters are mutually exclusive. Printer and class names must not exceed 14 characters and must consist entirely of the characters **A-Z**, **a-z**, **0-9** and _ (underscore).

Models

Model interface programs are supplied with the LP software. They are shell procedures, C programs, or other executable programs that interface between *lpsched*(1M) and devices. All printer models reside in directory **/usr/spool/lp/model** and can be used without modification with **lpadmin -m**. All cancel models reside in directory **/usr/spool/lp/cmodel** and can be used without modification with **lpadmin -ocm**. All status models reside in directory **/usr/spool/lp/smodel** and can be used without modification with **lpadmin -osm**. Models should have 644 permission if owned by **lp** and **bin**, or 664 permission if owned by **bin** and **bin**. Model file names must not exceed 14 characters. Alternatively, LP administrators can modify copies of models then use **lpadmin -m** to associate them with printers. See *mklp*(1M) for details of the printer models provided with your HP-UX system.

The LP model interface program does the actual printing on the device that is currently associated with the printer. The LP spooler sets standard input to **/dev/null** and standard output and standard error output to the device specified in the **-v** option of *lpadmin*. The interface program is then invoked for printer *P* from the directory **/usr/spool/lp** as follows:

interface/*P id user title copies options file* ...

where arguments are as follows:

id request id returned by *lp*(1).

user login name of the user who made the request.

title optional title specified with the **-t** option of *lp*(1).

copies number of copies to be printed.

options blank-separated list of class-dependent or printer-dependent options specified with the **-o** option of *lp*(1). Options from a BSD system have the character sequence **BSD** attached to the beginning of the option (for example, **BSDl**).

file full pathname of the file to be printed.

Given the command line arguments and the output directed to the device, interface programs can format their output in any way they choose.

When printing is completed, it is the responsibility of the interface program to exit with a code indicative of the success of the print job. Only return values of **0** indicating that the job completed successfully, or values of positive **1** through **127** indicating that some error was encountered that does not affect future print jobs should be used. Negative values and positive values greater than 127 are reserved for system use and should not be used by interface programs. *lpsched*(1M) notifies users by mail when there is an error in printing the request. If problems are detected that are likely to affect future print jobs, the

interface program should disable the printer so that other pending print requests are not lost.

The cancel and status model interface programs perform the actual communication with the remote system to cancel requests or get the status of requests. See *rcancel*(1M) and *rlpstat*(1M) for command line arguments.

HP Clustered Environment

In the HP Clustered Environment, all spooling is handled as if the cluster nodes were a single system and all printers attached to either the cluster server or clients can be available. Remote spooling applies to spooling from or to machines outside of the cluster nodes.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, *lpadmin* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

EXAMPLES

Assuming an existing Hewlett-Packard HP 2934A line printer named **lp1**, it will use the **hp2934a** model interface through **/dev/lp** after the command:

/usr/lib/lpadmin -plp1 -mhp2934a -v/dev/lp

Assuming an existing Hewlett-Packard HP 33440A laser printer locally attached to the cluster client **cluster1**, it will use the **hp33440a** model interface through **/dev/laser** which resides on **cluster1** after the command:

/usr/lib/lpadmin -plp2 -mhp33440a -v/dev/laser -acluster1

Assuming a printer **lp** on a remote system **system2**, the command:

/usr/lib/lpadmin -plp3 -v/dev/null -mrmodel -ocmrcmodel -osmrsmodel -ob3 -ormsystem2 -orplp -v/dev/null

causes the spool system to use the local line printer **lp3** and the model **rmodel**. The spool system also uses the model **rcmodel** to cancel remote requests and **rsmodel** to get status from **system2**. In addition, the three-digit sequence numbers, the remote system name **system2** and the remote printer **lp** are used.

WARNINGS

When installing remote printers, use the option **-ocmrcmodel** instead of **-oci/usr/lib/rcancel** to specify the method used to cancel remote requests. The option **-osmrsmodel** should be used instead of **-osi/usr/lib/rlpstat** to specify the method used for displaying remote status.

When installing printers attached to cluster clients, *lp* on the cluster server must be permitted to access the cluster clients using **remsh(1)** without supplying a passwd.

When installing printers attached to cluster clients, the device file associated with the printers must be accessed by processes on the cluster clients. See **mknod(1M)**.

*class*es must not include *remote* printers. HP-UX systems do not have the ability to distribute print jobs in this way. Printing to a class of printers on a remote system (**systemB** for example) must be accomplished by creating the class on the remote system, then identifying that class by using a command resembling:

lpadmin -plocal_name -ormsystemB -orosystemB_class_name

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1), lpstat(1), nroff(1), accept(1M), lpana(1M), lpsched(1M), mklp(1M), mknod(1M), rcancel(1M), rlp(1M), rlpdaemon(1M), rlpstat(1M).

NAME

lpana - print LP spooler performance analysis information

SYNOPSIS

`lpana` [`-d` *dest*]

DESCRIPTION

`lpana` prints information about the current performance of the LP line printer system for use by system administrators when determining how to configure the entire spooler system for optimum operation.

Options

`lpana` recognizes one option:

`-d` *dest* Choose *dest* as the printer or the class of printers. If *dest* is a printer, the performance analysis information is printed on that specific printer. If *dest* is a class of printers, the performance analysis information is printed on the printers that are members of the class. By default, `lpana` prints the performance analysis information for all printers and/or classes.

`lpana` examines `/usr/spool/lp/lpana.log` for the following items:

`Wait AV` Average waiting time from when job is spooled until start of printing.

`Wait SD` Standard Deviation for waiting time.

`Print AV` Average printing time from start to end of job.

`Print SD` Standard Deviation for printing time.

`Bytes AV` Average of number of bytes printed per request.

`Bytes SD` Standard Deviation for number of bytes.

`Sum KB` Sum of bytes printed for all requests (in Kbytes).

`Num of Requests`
Total number of requests since logging started.

HP Clustered Environment

In the HP Clustered Environment, all spooling is handled as if the cluster nodes were a single system and all printers attached to either the cluster server or clients can be available. Remote spooling applies to spooling from or to machines outside of the cluster nodes.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

`lpana` performs its operation on the local system (or HP cluster) only.

AUTHOR

`lpana` was developed by HP.

FILES

`/usr/spool/lp/lpana.log`

SEE ALSO

lp(1), lpstat(1), lpadmin(1M), lpsched(1M).

NAME

lpsched, lpshut, lpmove, lpfence - start/stop the LP request scheduler, move requests, and define the minimum priority for printing

SYNOPSIS

/usr/lib/lpsched [**-v**] [**-a**]
/usr/lib/lpshut
/usr/lib/lpmove *requests dest*
/usr/lib/lpmove *dest1 dest2*
/usr/lib/lpfence *printer fence*

DESCRIPTION

lpsched Schedules requests taken by *lp*(1) for printing on line printers. *lpsched*(1M) is typically invoked in **/etc/rc**. This creates a process which runs in the background until *lpshut* is executed. The activity of the process is recorded in **/usr/spool/lp/log**.

lpsched recognizes the following options:

-v Write a verbose record of the *lpsched* process on **/usr/spool/lp/log**.

-a Write *lpana(1M)* logging data on **/usr/spool/lp/lpana.log**.

lpshut

Shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked stop printing. Requests that were printing at the time a printer was shut down are reprinted in their entirety after *lpsched* is started again. All LP commands perform their functions even when *lpsched* is not running.

lpmove

Moves requests that were queued by *lp*(1) between LP destinations. This command can be used only when *lpsched* is not running.

The first form of the command moves the named *requests* to the LP destination, *dest*. *requests* are request ids as returned by *lp*(1). The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp*(1) rejects requests for *dest1*.

Note that *lpmove* never checks the acceptance status (see *accept*(1M)) for the new destination when moving requests.

lpfence

Defines the minimum required *priority* for the spooled file to be printed. *fence* must be in between 0 (lowest fence) and 7 (highest fence). Each *printer* has its own *fence*, which is initialized to 0 when it is configured by the *lpadmin*(1M) command. *lpfence* is used only when *lpsched* is not running.

HP Clustered Environment

In the HP Clustered Environment, all spooling is handled as if the cluster nodes were a single system and all printers attached to either the cluster server or clients can be available. Remote spooling applies to spooling from or to machines outside of the cluster nodes.

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings.

LANG determines the language in which messages are displayed.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *lpsched*, *lpmove*, and *lpshut* behave as if all internationalization variables are set to "C". See *environ*(5).

FILES

/usr/spool/lp/*

WARNINGS

Moving requests associated with remote printers can cause unpredictable results.

lpsched, *lpshut*, *lpmove*, and *lpfence* perform their operation on the local system (or HP cluster) only.

SEE ALSO

accept(1M), cancel(1), enable(1), lp(1), lpadmin(1M), lpana(1M), lpstat(1), rcancel(1M), rlp(1M), rlpdaemon(1M), rlpstat(1M).

NAME

lsdev - list device drivers in the system

SYNOPSIS

`/etc/lsdev` [*major* ...]

DESCRIPTION

`lsdev` lists, one pair per line, the major device numbers and driver names of device drivers configured into the system and available for invocation via special files. A `-1` in either the block or character column means that a major number does not exist for that type.

If no arguments are specified, `lsdev` lists all drivers configured into the system. If any arguments are used, they must represent major device numbers; `lsdev` then lists the corresponding driver name (if any).

`lsdev` is nothing more than a quick-reference aid. In some implementations, `lsdev` may only read an internal list of device drivers, rather than the actual list in the operating system.

DIAGNOSTICS

Lists the driver name as `no such driver` or `invalid major number` when appropriate.

WARNINGS

Some device drivers available from the system may be intended for use by other drivers. Trying to use them directly from a special file may produce unexpected results.

AUTHOR

`lsdev` was developed by HP.

DEPENDENCIES

Series 300/400:

An internal list, not the actual drivers in the operating system, is listed.

Series 700/800:

The actual list of drivers in the operating system is listed. In addition to the driver name, each line also lists the device class.

SEE ALSO

Section 7 entries related to specific device drivers.

NAME

lssf - list a special file

SYNOPSIS

lssf *special_file* ...

DESCRIPTION

lssf lists information about a special file. For each *special_file* name, **lssf** determines the major number of the special file and whether it is block or character (using **stat()**). It then scans the system for the device that is associated with the special file. When the device is found, the minor number of the special file is decoded. A mnemonic description of the minor number is printed on standard output along with the hardware path (i.e., address) of the device. Mnemonics used to describe the fields are closely related to the options used with **mksf** (see *mksf*(1M)).

DIAGNOSTICS

Most diagnostic messages from **lssf** are self explanatory. Listed below are some messages deserving further clarification. Warnings allow **lssf** to continue.

Warnings

No such device in the system
: There is no information about the device in the kernel. The special file is not usable. Use **rmsf** to remove the special file (see *rmsf*(1M)).

Character major <*major*> is not in the kernel
Block major <*major*> is not in the kernel
: The major number associated with the special file is not in the kernel. Use **uxgen** to add the appropriate driver to the kernel (see *uxgen*(1M)).

Device driver <*name*> is not in the kernel
Device class <*name*> is not in the kernel
: The indicated device driver or device class is not present in the kernel. An **open()** of a special file pointing to an unusable device fails. To make the device usable, the appropriate device driver and/or device class must be added to the **uxgen** input file and a new kernel generated (see *uxgen*(1M)). If the device is no longer needed, **rmsf** should be used to remove the special files and update **/etc/ioconfig**.

<*special_file*> is not a special file
: The file is not associated with an I/O device.

EXAMPLES

Suppose a special file is created with the command **mksf -d tape1 -H 8.6.1 -l 2 -b 1600 -a mt/2m**. The command **lssf mt/2m** then produces:

```
tape1 lu 2 bpi 1600 att address 8.6.1 mt/2m
```

AUTHOR

lssf was developed by HP.

FILES

/dev/config

SEE ALSO

insf(1M), mksf(1M), rmsf(1M), uxgen(1M).

NAME

lvchange - change logical volume characteristics

SYNOPSIS

/etc/lvchange [**-a** *availability*] [**-d** *schedule*] [**-p** *permission*] [**-r** *relocate*] [**-s** *strict*] [**-C** *contiguous*] [**-M** *mirror_write_cache*] [**-c** *mirror_consistency*] *lv_path*

DESCRIPTION

lvchange changes the characteristics of a logical volume. Optional command-line options and parameters specify the type and extent of change. Each current characteristic for a logical volume remains in effect until explicitly changed by the corresponding command option. All options take effect immediately except **-s** which takes effect only when new extents are allocated by the **lvextend** command (see *lvextend*(1M)). *lv_path* must be a logical volume name.

Options

lvchange recognizes the following options and parameters:

-a *availability* Set Logical Volume availability. *availability* can have one of the following values:

- **y** Make a logical volume available; that is, an open of the logical volume will succeed.
- **n** Make a logical volume temporarily unavailable; that is, an open of the logical volume will fail; however, all current processes that have the logical volume open remain open.

-d *schedule*
Set scheduling policy when a logical extent with more than one mirror is written. *schedule* can have one of the following values:

- **p** Establish a parallel scheduling policy.
- **s** Establish a sequential scheduling policy. Use this value with care because it leads to performance loss in most cases.

-p *permission*
Set access permission to read-write or read-only. *permission* can have one of the following values:

- **w** Set access permission to read-write.
- **r** Set access permission to read-only.

-r *relocate*
Set the bad block relocation policy. *relocate* can have one of the following values:

- **y** Allow bad block relocation.
- **n** Prevent bad block relocation.

-s *strict*
Specify the strict allocation policy. Mirrors of a logical extent can be allocated to share or not share the same physical volume or physical volume group. This option only makes sense when the physical volumes of the volume group that owns the logical volume to be changed reside on different physical disks. *strict* can have one of the following values:

- **y** Set a strict allocation policy: mirrors of a logical extent cannot share the same physical volume.
- **n** Do not set a strict or PVG-strict allocation policy.
- **g** Set a PVG-strict allocation policy: mirrors of a logical extent cannot share the same physical volume group.

When the logical volume is mirrored, the following changes are not allowed:

- from *non-strict* to *strict*,
- from *non-strict* to *PVG-strict*,
- from *strict* to *PVG-strict*.

`-C` *contiguous*
Specify the contiguous allocation policy. Physical extents are allocated in ascending order without any gap between adjacent extents and all extents are contained in a single physical volume. *contiguous* can have one of the following values:

`y` Set a contiguous allocaton policy.

`n` Do not set a contiguous allocation policy.

A non-empty logical volume that has a non-contiguous allocation policy cannot be changed to a contiguous allocaton policy unless it happens to meet all the requirements of the contiguous allocation policy. The strict (`-s`) and contiguous (`-C`) options can be used together to change two allocation policies at the same time.

See *lvcreate*(1m) for more information about contiguous allocation policy.

`-M` *mirror_write_cache*
Set the Mirror Write Cache to *on* or *off*. This option is allowed only when the logical volume is not opened. *mirror_write_cache* can have one of the following values:

`y` Set Mirror Write Cache to *on*. Every write to a mirror copy is recorded in the Mirror Write Cache and written into the Mirror Consistency Record on the disk if a cache-miss occurred. This allows LVM to determine whether all mirror copies are identical, even across system crashes. When the volume group is activated, the Mirror Consistency Record is used to perform mirror consistency recovery.

`n` Set Mirror Write Cache to *off*. Mirror write does not incur an additional write to the Mirror Consistency Record on the disk.

`-c` *mirror_consistency*
Set mirror consistency recovery to *on* or *off*. This option only makes sense when `-M n` is specified, or when the Mirror Write Cache has already been turned off on the logical volume. *mirror_consistency* can have one of the following values:

`y` Set mirror consistency recovery to *on*. LVM achieves mirror consistency during volume group activation by going through all logical extents and copying data from a *non-stale* copy to the other mirror copy.

`n` Set mirror consistency recovery to *off*. LVM does not perform mirror consistency recovery on this logical volume when the volume group is activated.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **lvchange** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Change the permission of a logical volume to read-only:

```
lvchange -p r /dev/vg01/lvol3
```

Change the allocation policy of a logical volume to non-strict:

```
lvchange -s n /dev/vg01/lvol7
```

Turn the mirror write cache off on a logical volume:

```
lvchange -M n /dev/vg01/lvol1
```

REMARKS

The `-r` option is not available on HP-IB devices.

SEE ALSO

lvcreate(1M), lvdisplay(1M), lvextend(1M).

NAME

lvcreate - create a logical volume in a volume group

SYNOPSIS

/etc/lvcreate [**-d** *schedule*] {**-l** *logical_extents_number* | -L *logical_volume_size*} [**-m** *mirror_copies*] [**-n** *lv_path*] [**-p** *permission*] [**-r** *relocate*] [**-s** *strict*] [**-C** *contiguous*] [**-M** *mirror_write_cache*] [**-c** *vol_group_name*

DESCRIPTION

lvcreate creates a new logical volume within the volume group specified by *vol_group_name*.

If *lv_path* is specifed on the command line, a new logical volume is created with that name. If *lv_path* is not specified, a system-generated name of the form **lvol***N* is created, where *N* is the minor number of the new logical volume starting with minor number 1.

Default settings provide the most commonly used characteristics, but use options to tailor the logical volume to the requirements of the system. Once a logical volume is created, its characteristics can be changed by using the **lvchange**, **lvextend**, and **lvreduce** commands (see *lvchange*(1M), *lvextend*(1M), and *lvreduce*(1M)).

-d *schedule* Set scheduling policy when one logical extent with more than one mirror is written. *schedule* can have one of the following values:

p (default) Establish a parallel scheduling policy.

s Establish a sequential scheduling policy. This value should be used with care since, in most cases, it leads to performance loss.

-l *logical_extents_number*
Allocate the number of logical extents specified by *logical_extents_number* to the *lv_path*. *logical_extents_number* is a numeric value ranging from **1** through **65535** (implemtation limit); default value is zero. Either this option or the **-L** option can be specificed, but not both.

-L *logical_volume_size*
Create a logical volume of size *logical_volume_size*, where *logical_volume_size* is specified in units of Mbytes. The maximum value for *logical_volume_size* is **4096** Mbytes (4 Gbytes).

Allocate the number of logical extents corresponding to the *logical_volume_size*. If the *logical_volume_size* is not a multiple of extent size, the size is adjusted to be the next multiple. The number of logical extents can be a value ranging from 1 through 65 535.

Either this option or the **-l** option can be specified, but not both.

-m *mirror_copies*
Set the number of physical extents (mirrors) allocated for each logical extent.

mirror_copies specifies the number of mirror copies that contain the same data as the original; value can be **1** or **2**, meaning, respectively, one or two mirror copies that contain the same data as the original. Default value for *mirror_copies* is zero.

This option requires installation of optional LVM MIRRORING software (not included in the standard HP-UX operating system) before it can be used.

-n *lv_path*
Create a new logical volume with the specified name *lv_path* where *lv_path* is a simple file name, not a path name.

-p *permission*
Set the access permission to read-write or read-only. *permission* can have one of the following values:

w (default) Set access permission to read-write.

r Set access permission to read-only.

-r *relocate*
Set bad block relocation policy. *relocate* can have one of the following values:

y (default) Cause bad block relocation to occur.

`n` Prevent bad block relocation from occurring.

`-s` *strict*
Define strict allocation policy. Mirror copies of a logical extent can be allocated to share or not share the same physical volume (or physical volume group). This option only makes sense when the physical volumes (of the volume group that owns the logical volume to be changed) reside on different physical disks. *strict* can have one of the following values:

`y` (default) Set a strict allocation policy; mirrors for a logical extent cannot share the same physical volume.

`n` Do not set a strict allocation policy; mirrors for a logical extent can share the same physical volume.

`g` Set a PVG-strict allocation policy; mirrors for a logical extend cannot share the same physical volume group. PVG-strict allocation policy cannot be set on a logical volume in a volume group that does not have any physical volume group defined.

`-C` *contiguous*
Defines the contiguous allocation policy. A contiguous logical volume has three characteristics:

- Physical extents are allocated in ascending order,
- No gap is allowed between physical extents within a mirror copy,
- Physical extents of any mirror copy all reside in a single physical volume.

Use the strict (`-s`) and contiguous (`-C`) options together to form various combined allocation policies on a logical volume. For example, `-s y -C y` means to create a logical volume such that each mirror copy is contiguous, yet mirror copies of a logical extent cannot share the same physical volume.

`y` Set a contiguous allocation policy.

`n` (default) Do not set a contiguous allocation policy.

`-M` *mirror_write_cache*
Set Mirror Write Cache to on or off. *mirror_write_cache* can have one of the following values:

`y` (default) Set Mirror Write Cache to on. Every write to a mirror copy is recorded in the Mirror Write Cache and written to the Mirror Consistency Record under the Volume Group Reserved Area on the disk. This allows the LVM to determine whether all the mirror copies are identical, even across system crashes. When the volume group is activated, the Mirror Consistency Record is used to perform mirror consistency recovery.

`n` Set Mirror Write Cache to off. Mirror write does not incur an additional write to the Mirror Consistency Record.

`-c` *mirror_consistency*
Set mirror consistency recovery to on or off. This option only makes sense when `-M n` is specified. *mirror_consistency* can have one of the following values:

`y` (default) Set mirror consistency recovery to on. LVM achieves mirror consistency during volume group activation by going through all logical extents and copying data from a **non-stale** copy to the other mirror copies.

`n` Set mirror consistency recovery to off. LVM does not perform mirror consistency recovery on this logical volume when the volume group is activated.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, `lvcreate` behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Create a logical volume in volume group `/dev/vg02`:

```
lvcreate /dev/vg02
```

Create a logical volume in volume group `/dev/vg03` with non-strict allocation policy:

```
lvcreate -s n /dev/vg03
```

Create a logical volume of size 100 Mbytes in volume group `/dev/vg03`:

```
lvcreate -L 100 /dev/vg03
```

REMARKS

The `-m` and `-r` options cannot be used with HP-IB devices.

SEE ALSO

lvchange(1M), lvextend(1M), lvreduce(1M), pvchange(1M).

(Series 800 Only)

NAME

lvdisplay - display information about logical volumes

SYNOPSIS

`lvdisplay` [`-v`] *lv_path* ...

DESCRIPTION

`lvdisplay` displays the characteristics and status of each logical volume specified by the *lv_path* parameter. If the `-v` (verbose) option is specified, `lvdisplay` displays physical volume (PV) distribution information, and map information about the physical extents that correspond to the logical extents of the logical volume or volumes.

Command with no Options Specified

If the `-v` option is not specified, `lvdisplay` displays the following operating information:

`LV Name`: Name of the logical volume

`VG Name`: Name of the volume group

`LV Permission`:
Access permission: read-only or read-write

`LV Status`: State of the logical volume:

`available/stale`
Logical volume is available but contains physical extents that are not current.

`available/syncd`
Logical volume is available and synchronized.

`available`, Logical volume is available but the stale/syncd state cannot be confidently determined because the logical volumes have both the Mirror Write Cache and Mirror Consistency Recovery turned off.

`unavailable`
Logical volume is not available for use.

`Mirror Copies`:
Number of physical extents beyond the original allocated for each logical extent; i.e., the number of mirrors: 1, 2 or 3.

`Consistency Recovery`:
Mode of mirror consistency recovery which determines how LVM performs mirror consistency recovery during volume group activation:

`MWC` Recover mirror consistency by using the Mirror Write Cache and Mirror Consistency Record. Implies that Mirror Write Cache is on.

`NOMWC` Recover mirror consistency by going through all logical extents and copying data from a *non-stale* copy to the other mirror copies. Implies that Mirror Write Cache is off.

`NONE` No mirror consistency recovery during volume group activation on this logical volume. Implies that Mirror Write Cache is off.

`Schedule`:
Sequential or parallel scheduling policy

`LV Size`:
Size of the logical volume in Mbytes.

`Current LE`:
Number of logical extents currently in the logical volume

`Allocated PE`:
Number of physical extents allocated to the logical volume

`Bad Blocks`:
Bad block relocation policy

`Allocation:`
Current allocation state:

`non-strict`	`non-strict/contiguous`
`strict`	`strict/contiguous`
`PVG-strict`	`PVG-strict/contiguous`

`strict` allocation specifies that mirror copies for a logical extent are not allocated on the same physical volume.

`PVG-strict` allocation specifies that mirror copies for a logical extent are not allocated on the same physical volume group.

`non-strict` allocation specifies that physical extents that belong to the same logical extent can be allocated on the same physical volume or physical volume group.

`contiguous` allocation specifies that physical extents are allocated in an ascending order without any gap between adjacent extents and that all physical extents of a given mirror are contained in a single physical volume.

Command with -v Option Specified
When the `-v` option is specified, `lvdisplay` lists the map and additional information about the distribution of the logical volume across the physical volumes of the volume group.

`Distribution of logical volume:`
Lists distribution of logical volume *lv_path* within the volume group.

`PV Name:` Name of the physical volume to which the logical extents are allocated

`LE on PV:` Number of logical extents allocated on the physical volume

`PE on PV:` Number of physical extents allocated on the physical volume

`Logical extents:`
Displays the following information for each logical extent:

`LE:` Logical extent number

`PV1:` Physical volume name that corresponds to the location of the first physical extent of the logical extent

`PE1:` First physical extent number allocated to the logical extent

`Status 1:` Status of the first physical extent: `stale` or `current`

`PV2:` The physical volume name that corresponds to the location of the second physical extent (first copy) of the logical extent

`PE2:` Second physical extent number allocated to the logical extent

`Status 2:` Status of the second physical extent: `stale` or `current`

`PV3:` Physical volume name that corresponds to the location of the third physical extent (second copy) of the logical extent

`PE3:` Third physical extent number allocated to the logical extent

`Status 3:` Status of the third physical extent: `stale` or `current`

EXTERNAL INFLUENCES
Environment Variables
`LANG` determines the language in which messages are displayed.

If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`.

If any internationalization variable contains an invalid setting, `lvdisplay` behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES
Display information about a logical volume:

```
lvdisplay /dev/vg01/lv03
```

Display all the available information about a logical volume, including the characteristics, status and distribution map:

```
lvdisplay -v /dev/vg01/lv03
```

SEE ALSO

lvchange(1M), lvcreate(1M), lvextend(1M), lvreduce(1M), pvdisplay(1M), vgdisplay(1M).

NAME

lvextend - increase number of physical extents allocated to a logical volume

SYNOPSIS

/etc/lvextend {**-l** *logical_extents_number* | **-L** *logical_volume_size* | **-m** *mirror_copies*} *lv_path* [*physical_volume_path* ... | *physical_vol_group_name*...]

DESCRIPTION

lvextend increases the number of mirrored copies or the size of the *lv_path* parameter. The change is determined according to which command options are specified.

To limit the allocation to specific physical volumes, use the names of one or more physical volumes in the *physical_volume_path* parameter or use the names of one or more physical volume groups in the *physical_vol_group_name* parameter; otherwise, all of the physical volumes in a volume group are available for allocating new physical extents. LVM always ensures that physical extent allocation can satisfy the current allocation policy or policies. If a physical volume is not suitable for use with a certain allocation policy, it is not used during physical extent allocation, even it is specified in the *physical_volume_path* parameter or indirectly in the *physical_vol_group_name* parameter.

The *physical_vol_group_name* parameter is allowed only if one of the allocation policies of the logical volume is PVG-strict.

Options

lvextend recognizes the following options and accompanying arguments:

-l *logical_extents_number*

Increases the number of logical extents allocated to *lv_path*. *logical_extents_number* must be greater than the number of logical extents previously allocated to *lv_path* and less than 65 535 which is the implementation limit.

logical_extents_number represents the new total number of logical extents that can be allocated to *lv_path*. The change is accomplished by allocating the number of additional logical extents represented by the difference between *logical_extents_number* and the previous number of extents.

The mirror policy and mirror copies number for the new logical extents is the same as previously established for *lv_path*.

One, and only one of the **-l**, **-L**, or **-m** must be supplied.

-L *logical_volume_size*

Increases the size of logical volume to *logical_volume_size*, where *logical_volume_size* is specified in units of Mbytes. The maximum value for *logical_volume_size* is **4096** Mbytes (4 Gbytes). If *logical_volume_size* is not a multiple of extent size, the size is adjusted to the next multiple.

The **-L** option is same as the **-l** option, where the *logical_extents_number* is equivalent to *logical_volume_size* divided by the extent size.

See the description of **-l** above.

-m *mirror_copies*

Sets the number of physical extents allocated for each logical extent.

mirror_copies (that is, mirrors) can be either **1** or **2**. This means that beyond the original copy, one or two mirror copies always contain the same data as the original copy.

mirror_copies must be greater than the current number of mirrors for the logical volume.

Data in the new copies is synchronized. The synchronization process can be time consuming, depending on hardware characteristics and the amount of data.

This option requires installation of optional LVM MIRRORING software (not included in the standard HP-UX operating system) before it can be used.

One, and only one of the options `-l`, `-L`, or `-m` must be specified.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **lvextend** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Increase the number of the logical extents of a logical volume to one hundred:

```
lvextend -l 100 /dev/vg01/lvol3
```

Increase the logical volume size to 400 Mbytes:

```
lvextend -L 400 /dev/vg01/lvol4
```

Allocate two mirrors (that is, three copies) for each logical extent of a logical volume:

```
lvextend -m 2 /dev/vg01/lvol5
```

WARNINGS

The `-m` option cannot be used on HP-IB devices.

SEE ALSO

lvcreate(1M), lvdisplay(1M), lvreduce(1M), pvchange(1M), pvdisplay(1M).

NAME

lvlnboot - prepare a Logical Volume to be a root, primary swap, or dump volume

SYNOPSIS

/etc/lvlnboot [**-v**][**-r** *root_lv*][**-s** *swap_lv*][**-d** *dump_lv*][**-R** *volume_group_name*]

DESCRIPTION

lvlnboot updates all Physical Volumes in the Volume Group such that the Logical Volume becomes the root, primary swap, or a dump volume when the system is next booted on the Volume Group. If a non-existent logical volume is specified, this command fails. If a different logical volume is already linked to the root or primary swap, the command fails.

If the length or mirrored copies of a root, swap, or dump logical volume changes via the **lvextend** command (see *lvextend*(1M)), **lvlnboot** must be run again on each logical volume that has changed.

Options

lvlnboot recognizes the following options and arguments:

-v Prints verbose messages. With no other arguments present, this option prints information on all Volume Groups.

-r*root_lv* Defines *root_lv* to be the root volume the next time the system is booted on this Volume Group. Updates the **Boot Data Reserved Area** such that the Volume Group is used to locate the root file system. This allows the *root_lv* to be used as the root volume during a maintenance mode boot. The Physical Volume or volumes containing the *root_lv* must have been created using the **pbcreate -B** option (see *pvcreate*(1M)), indicating that that Physical Volume is to be used as a Bootable Physical Volume. Also, the *mkboot(1M)* command must have been run on the Physical Volume to create the LIF area at the top of the Physical Volume. The *root_lv* must be a contiguous Logical Volume and cannot have Bad Block Relocation enabled.

-s*swap_lv* Defines *swap_lv* to be the primary swap volume next time the system is booted on the Volume Group. Updates the **Boot Data Reserved Area**. Any existing swap areas previously defined are removed. *swap_lv* must be a contiguous Logical Volume, and a Root Logical Volume must have been previously defined by use of this command.

-d*dump_lv* Defines *dump_lv* to be one of the dump volumes next time the system is booted on the Volume Group. Updates the **Boot Data Reserved Area**. The combined size of all the dump volume should be atleast 2048 bytes larger than the total memory of the system. The additional 2 Kbytes is used to safeguard against dump to the bottom of the disk. Multiple dump devices can be configured, but each *dump_lv* must be entered on a separate **lvlnboot** command line. *dump_lv* must be an unmirrored, contiguous logical volume.

-R*volume_group_name*
Recovers any missing links to all of the logical volumes specified in the **Boot Data Reserved Area** on each of the Physical Volumes in the Volume Group.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **lvlnboot** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Specify that logical volume **/dev/vg00/lvol3** should be used as the Root Logical Volume:

```
lvlnboot -r /dev/vg00/lvol3
```

Specify that logical volume **/dev/vg00/swap1, /dev/vg00/dump2,** and **/dev/vg00/dump3,** should be used as the Dump Logical Volumes and that **/dev/vg00/swap1** should also be used as Primary Swap:

```
lvlnboot -s /dev/vg00/swap1
lvlnboot -d /dev/vg00/swap1
lvlnboot -d /dev/vg00/dump2
lvlnboot -d /dev/vg00/dump3
```

SEE ALSO

lvrmboot(1M), pvcreate(1M), mkboot(1M).

NAME

lvmerge - merge two logical volumes into one logical volume

SYNOPSIS

`/etc/lvmerge` *backup_lv_path master_lv_path*

Remarks:

This command requires installation of optional LVM MIRRORING software (not included in the standard HP-UX operating system) before it can be used.

DESCRIPTION

`lvmerge` merges two Logical Volumes of the same size, increasing the number of mirrored copies of the *master_lv_path* by the number of copies in the *backup_lv_path*.

Data previously contained in the *backup_lv_path* is resynchronized using the data in the *master_lv_path*. All new data on the *backup_lv_path* is lost.

Whenever a mirrored logical volume is split into two logical volumes, a bitmap is stored that keeps track of all writes to either logical volume in the split pair. When the two logical volumes are subsequently merged using `lvmerge`, the bitmap is used to decide which areas of the logical volumes need to be resynchronized. This bitmap continues to exist until the merge is completed, or until either of the logical volumes is extended or reduced, or the system is rebooted.

If there is no bitmap available, the entire logical volume is resynchronized.

The normal usage for this command is to merge previously mirrored logical volumes that have been split using the `lvsplit` command (see *lvsplit*(1M). However, the two logical volumes are not required to have been the result of a previous `lvsplit` operation.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, `lvmerge` behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Merge `/dev/vg00/lvol1b` with `/dev/vg00/lvol1`:

```
lvmerge /dev/vg00/lvol1b /dev/vg00/lvol1
```

WARNINGS

All data on *backup_lv_path* is lost after the merge if no bitmap is found.

`lvmerge` does not check to guarantee that the allocation policy of *master_lv_path* is preserved after the merge.

SEE ALSO

lvsplit(1M), lvcreate(1M), lvextend(1M).

NAME

lvmmigrate - prepare root file system for migration from partitions to logical volumes

SYNOPSIS

`/etc/lvmmigrate [-v] [-n] [-d` *disk_special_file* `] [-f] [-i` *file_system* ... `] [-e` *file_system* ...]

DESCRIPTION

`lvmmigrate` records the configuration information of the current system in the LIF volume of the boot section (6) for use with the `install` command (see *install*(1M)). If there is no LIF volume on the disk, `lvmmigrate` creates it using *lifinit*(1), then records the information in a LIF file named `LVMMIGRATE`. The information is also written to file `/tmp/LVMMIGRATE` for reviewing. `install` looks the for the LIF file `LVMMIGRATE`, and if it exists, uses the information found as the configuration defaults for the root volume group and the root file systems. After `install` has completed, a copy of the `LVMMIGRATE` file can be found on the newly created system in the file `/local/LVMMIGRATE`.

All file system entries in the `/etc/mnttab` and `/etc/checklist` files are read. `lvmmigrate` also searches for unmounted file systems and possible character data sections in unused disk areas. The list of file systems appropriate for the `root volume group` are marked for migration. They are `/`, `/tmp`, `/usr`, and any file system with mount path beginning as `/usr/`.

`lvmmigrate` displays the following information on the standard output: disks and file system names that are marked for migration, disk areas and file systems to be backed up by the user, and instructions for reinstallation.

After execution of `lvmmigrate`, users *must* backup the file systems and any character device sections to tape. The system is then reinstalled on logical volumes using the configuration information recorded by `lvmmigrate`.

Options

`lvmmigrate` recognizes the following options:

`-v` (verbose) `lvmmigrate` displays all disks, file systems, possible raw sections, and tape devices present in the system.

`-n` Specifies a "no write" operation for preview purposes. Migration information is displayed on the terminal screen, but is not recorded in the boot section of the disk.

`-d` *disk_special_file*
Specifies the root disk to be used for reinstallation. Without this option the current root disk (where root file system `/` is currently located) is assumed and the configuration is recorded in the boot section.

`-f` Forces recording of configuration information. Information is recorded in a LIF file `LVMMIGRATE` in the boot section. Without this option, if there is a file system or LVM record in the boot section, no write is done and a warning message is displayed.

`-i` *file_system* ...
Specifies a list of file systems to be included in the `root volume group` along with `/`, `/tmp`, and `usr` file systems.

`-e` *file*`system ...`
Specifies a list of file systems to be excluded from the `root volume group`. Note that the `/` file system cannot be excluded.

EXAMPLES

Prepare a system for migration to root logical volumes. Create a file in the LIF area that `/etc/install` can use to read default configuration information. Specify verbose mode. Create file `/tmp/LVMMIGRATE`:

```
lvmmigrate -v
```

Display a detailed list of disks, file systems, and possible raw data sections present in the current system. The `-n` option disables writing of the `LVMMIGRATE` LIF file, but file `/tmp/LVMMIGRATE` is still created:

```
lvmmigrate -v -n
```

Include file system `/mnt` in the root volume group for migration and exclude file system `/usr/source`. Write configuration information in the boot section of disk `/dev/dsk/c1d0s2`:

```
lvmmigrate -d /dev/dsk/c1d0s2 -i /mnt -e /usr/source
```

WARNINGS

Use of the `-f` option results in overwriting the contents of the boot section. Before using the `-f` option be sure to backup all data on the boot section of the disk specified with the `-d` option.

If there is no LIF volume, `lvmmigrate` uses `lifinit` to create it (see *lifinit*(1). If file `LVMMIGRATE` already exists in the LIF volume, `lvmmigrate` rewrites it.

All data on disks being used for reinstallation must be backed up to a separate device because the install process overwrites data on all disks used in the new `root volume group`.

SEE ALSO

lifinit(1), install(1M).

NAME

lvreduce - decrease number of physical extents allocated to a logical volume

SYNOPSIS

`/etc/lvreduce` {`-m` *mirror_copies* | `-l` *logical_extents_number* | `-L` *logical_volume_size*} [`-f`] *lv_path* [*physical_volume_path* ...]

DESCRIPTION

`lvreduce` changes either the number of logical extents allocated to a logical volume specified by *lv_path*, or the number of physical extents allocated to each logical extent in the logical volume. The change is determined according to which command options you use. `lvreduce` removes mirror copies from *physical_volume_path*, if specified.

Options

`lvreduce` recognizes the following options and arguments:

`-f`
: Force reduction of the number of logical extents without first requesting confirmation. *Note:* This option can be dangerous when there is a file system on the *lv_path* that is larger than the size which the Logical Volume is being reduced to. If the file system is unmounted, the `-f` option forces the reduction the Logical Volume without reducing the file system. The file system becomes corrupt and is not mountable. If the file system is mounted, `lvreduce` fails, preventing a mounted file system from becoming corrupted. This option is allowed only if `-l` or `-L` is also specified.

`-m` *mirror_copies*
: Set the number of physical extents (mirrors) allocated for each logical extent.

  mirror_copies can have the value `0` or `1`. This means that beyond the original copy, no other (0) or one (1) mirror copy contains the same data as the original copy. If they are specified in the command line, mirror copies residing in *physical_volume_path* are used for `lvreduce`.

  mirror_copies must be less than the current number of mirrors for the logical volume. One, and only one of the three options `-m`, `-l`, and `-L` must be specified.

  This option requires installation of optional LVM MIRRORING software (not included in the standard HP-UX operating system) before it can be used.

`-l` *logical_extents_number*
: Decrease the number of logical extents allocated to the *lv_path*. *logical_extents_number* must be less than the number of logical extents previously allocated to *lv_path*.

  logical_extents_number represents the new total number of logical extents within *lv_path*. The change is accomplished by deallocating the number of logical extents represented by the difference between the previous number of extents and *logical_extents_number*.

  `lvreduce` asks for confirmation if the `-f` option is not specified.

  `lvreduce` requires use of one of the three options, `-l`, `-L`, or `-m`.

`-L` *logical_volume_size*
: Decrease the size of logical volume to *logical_volume_size*, where *logical_volume_size* is specified in units of Mbytes. If the *logical_volume_size* is not a multiple of extent size, the size is adjusted to the next multiple.

  The `-L` option is the same as `-l` option, where *logical_extents_number* is equivalent to *logical_volume_size* divided by extent size.

  See the description of `-l` option above.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, `lvreduce` behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Decrease the number of the logical extents of a logical volume to one hundred:

```
lvreduce -l 100 /dev/vg01/lvol3
```

Reduce to one mirror (that is, two copies) for each logical extent of a logical volume:

```
lvreduce -m 1 /dev/vg01/lvol5
```

Remove mirror copies of logical extents of a logical volume, possibly from the physical volume `/dev/dsk/c1d0s2`:

```
lvreduce -m 0 /dev/vg01/lvol4 /dev/dsk/c1d0s2
```

WARNINGS

The Logical Volume Manager does not store any information about which physical extents contain useful data; therefore, using the `-l` option might lead to the loss of useful data. Specifically, a file system corruption occurs if the *lvreduce* operation occurs on an unmounted file system. The *lvreduce* command on a Logical Volume containing a file system of greater length than the size being reducing to is *not* recommended.

To reduce a Logical Volume being used for swap, that swap area must be currently be in use.

SEE ALSO

lvcreate(1M), lvdisplay(1M), lvextend(1M), pvchange(1M), pvdisplay(1M).

NAME

lvremove - remove one or more logical volumes from a volume group

SYNOPSIS

`/etc/lvremove` [`-f`] *lv_path* ...

DESCRIPTION

`lvremove` removes the logical volume or volumes specified by *lv_path*. Logical volumes must be closed before they can be removed. For example, if the logical volume contains a file system, unmount the file system before removing it.

Options

`lvremove` recognizes the following option:

`-f` Specifies that no user confirmation is required.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, `lvremove` behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Remove a logical volume without requiring user confirmation:

```
lvremove -f /dev/vg01/lvol5
```

WARNINGS

This command destroys all data in the logical volume or volumes.

SEE ALSO

lvchange(1M), umount(1M).

NAME

lvrmboot - remove Logical Volume link to root, primary swap, or dump volume

SYNOPSIS

`/etc/lvrmboot` [`-v`][`-r`][`-s`][`-d` *dump_lv*] *volume_group_name*

DESCRIPTION

`lvrmboot` updates all Physical Volumes contained in the Volume Group such that the Logical Volume is removed as the root, primary swap, or a dump volume when the system is next booted on the Volume Group.

Options

`lvrmboot` recognizes the following options:

`-v` Prints verbose messages.

`-r` Remove definitions of the root, primary swap, and all dump volumes from the given Volume Group. Update the **Boot Data Reserved Area**.

`-s` Remove definition of the primary swap volume and all dump volumes from the given Volume Group. Update the **Boot Data Reserved Area**.

`-d`*dump_lv* Remove definition of *dump_lv* as one of the dump volumes. Update the **Boot Data Reserved Area**.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, `lvrmboot` behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Specify that the logical volume `/dev/vg00/lvol3` should be removed as one of the Dump Logical Volumes:

```
lvrmboot -v -d /dev/vg00/lvol3
```

Specify that volume group `/dev/vg00` should no longer be a Root Volume Group:

```
lvrmboot -r /dev/vg00
```

SEE ALSO

lvlnboot(1M).

NAME

lvsplit - split a mirrored logical volume into two logical volumes

SYNOPSIS

`/etc/lvsplit` [`-s` *suffix*] *lv_path*

Remarks:

This command requires installation of optional LVM MIRRORING software (not included in the standard HP-UX operating system) before it can be used.

DESCRIPTION

`lvsplit` splits singly- or doubly-mirrored *lv_path* into two logical volumes. A new logical volume is created containing one copy of the data.

If *suffix* is specified, the new logical volume is given the name *lv_pathsuffix*. If *suffix* is not specified, `lvsplit` assigns a new name using the suffix `b`.

Whenever a mirrored logical volume is split into two logical volumes, a bitmap is stored that keeps track of all writes to either logical volume in the split pair. When the two logical volumes are subsequently merged using `lvmerge`, the bitmap is used to decide which areas of the logical volumes need to be resynchronized (see *lvmerge*(1M)). This bitmap remains in existence until the merge is completed, or until either of the logical volumes is extended, reduced, or split again or the system is rebooted.

The new Logical Volume must be fs checked before mounting (see *fsck*(1M)). `lvsplit` flushs the filesystem to a consistent state except for pipes and unlinked but open files.

To remerge two mirrored copies of a Logical Volume, use the `lvmerge` command (see *lvmerge*(1M)).

EXTERNAL INFLUENCES

Environment Variables

`LANG` determines the language in which messages are displayed.

If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`.

If any internationalization variable contains an invalid setting, `lvsplit` behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Split the mirrored logical volume `/dev/vg00/lvol1` into two copies. Call the new logical volume `/dev/vg00/lvol1backup`:

```
lvsplit -s backup /dev/vg00/lvol1
```

Split an online Logical Volume which is currently mounted on */usr* so that a backup can take place:

```
lvsplit /dev/vg00/lvol1
fsck /dev/vg00/lvol1b
mount /dev/vg00/lvol1b /usr.backup
```

Perform backup operation:

```
umount /usr.backup
lvmerge /dev/vg00/lvol1b /dev/vg00/lvol1
```

SEE ALSO

lvmerge(1M), lvcreate(1M), lvextend(1M).

NAME

lvsync - synchronize stale logical volume mirrors in logical volumes

SYNOPSIS

`/etc/lvsync` *lv_path* ...

REMARKS

This software requires installation of optional LVM MIRRORING software (not included in the standard HP-UX operating system) before it can be used.

DESCRIPTION

`lvsync` synchronizes the physical extents of the logical volume specified by *lv_path*. Synchronization occurs only on physical extents that are **stale** mirrors of the original logical extent. The synchronization process can be time consuming, depending on the hardware characteristics and the amount of data.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, `lvsync` behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Synchronize the mirrors on a logical volume:

```
lvsync /dev/vg01/lvol5
```

SEE ALSO

lvdisplay(1M), vgsync(1M).

NAME

makecdf - create context-dependent files

SYNOPSIS

makecdf [[**-d** *default*] [**-f** *source_file*]] [[**-c** *context*] ...] *file* ...

DESCRIPTION

makecdf creates a context-dependent file (CDF) for each *file* passed on the command line. A list of contexts is generated from the **-c** options, or if no contexts are specified, *makecdf* creates a default context list from the cnode names appearing in **/etc/clusterconf**. CDF elements are created for each name in this context list (except as described below).

If *file* exists but is neither a CDF nor a directory, it is converted to a CDF whose elements are created by duplicating the original contents of *file* (or the contents of the *source_file* if the **-f** option is used).

If *file* exists and is an ordinary directory, the elements are not created by duplicating its contents into each CDF element. Instead, the files in the directory are moved into a directory whose name is the first context element mentioned on the command line, and empty directories are created for the remaining context elements. If no elements are specified (that is, if there are no **-c** options), the first entry in **/etc/clusterconf** is used as the name. In order to do this, the user must have write permission on *file* or be the super-user.

If *file* exists and is already a CDF, either the **-d** or **-f** option must be used to specify what should be duplicated to create the new CDF elements in the context list.

If *file* does not exist and there is no **-f** option, a CDF is created with empty contents whose names are those in the specified context list. In this case, CDF elements are created only for those contexts explicitly specified by the **-c** options (that is, **/etc/clusterconf** is not consulted).

When making a CDF out of a device file, *makecdf* makes an appropriate cnode-specific device file for each element named by cnode name if cnode name exists in **/etc/clusterconf**. Otherwise, the cnode ID of the original device file is copied to the new file.

Type, ownership, file mode bits, and access control lists of the CDF elements match those of *file* (see *acl*(5)). Contents of regular files and directories are copied to the new elements (see WARNINGS below). Ownership and the access control list of the CDF match those of the parent directory of *file*.

Only the file owner or superuser may convert a file to a CDF.

Options

makecdf supports the following options:

-c *context* Create the named CDF element. Causes *file*+/*context* to be created. More than one **-c** option can be specified to construct a *context* list.

-d default Duplicate an existing CDF element. Useful only for adding elements to existing context-dependent files. The **-d** option specifies which existing element of the CDF is to be duplicated into a new element.

-f *source_file* Duplicate *source_file* to the elements of the CDF. This option is only effective when *file* does not exist or is already a CDF. Ownership and permissions are preserved. The filename - indicates that the standard input should be copied to the elements, which become regular files with ownership and permissions much like files normally created by the shells. Note: The WARNING about file types below applies to this option as well as to *file* arguments.

RETURN VALUE

An exit code of **0** is returned if the CDFs are created without error. An exit code of **1** is returned in the case of any failure to create a CDF.

Warnings do not result in an exit code of **1**, because they are not considered catastrophic.

DIAGNOSTICS

Errors in system calls are displayed with whatever useful arguments are available.

Warnings are printed if operations cannot be performed as expected, although the CDF is still created. These warnings generally result from failure to change owner or change permissions of a file, or from inability to clean up the temporary file or fully recover from a failure.

Many messages make sense only when you understand how the program works. In particular, a temporary file is used to hold *file* while the CDF is created. The temporary file is then renamed, or copied and later removed. Some messages refer to operations attempted on the temporary file.

Most other messages are self explanatory.

EXAMPLES

Turn the file **/etc/issue** into a CDF with an element for each cluster node name in **/etc/clusterconf**:

makecdf /etc/issue

Turn the file **/etc/motd** into a CDF with the context elements **localroot** and **remoteroot**:

makecdf -c localroot -c remoteroot /etc/motd

Add the context **cnode3** to the existing CDF **/etc/issue** using the context element **cnode1** as the file to duplicate:

makecdf -c cnode3 -d cnode1 /etc/issue

Create a new CDF **menu** for which there is no existing file, copy the contents of **menu1** into the element **cnode1** and the contents of **menu2** into the element **cnode2**:

makecdf -c cnode1 -f menu1 menu
makecdf -c cnode2 -f menu2 menu

To move all files currently in **~/bin** to **~/bin+/HP-MC68010** while creating the empty directory **~/bin+/HP-MC68020**, execute the command:

makecdf -c HP-MC68010 -c HP-MC68020 ~/bin

WARNINGS

Note that *makecdf* attempts to create elements identical in type to *file*; thus, specifying the file **/dev/null** results in creating special file elements rather than creating empty regular files. The latter operation is correctly performed by using the command:

makecdf -f - file < /dev/null

makecdf treats directories somewhat differently than other types of files. (See the last example in the **EXAMPLES** section.)

makecdf does not overwrite existing elements of existing CDFs.

It is not intended that *makecdf* be used for creating context-dependent HP-UX system files. System files are created automatically during system installation or update.

DEPENDENCIES

NFS Access control lists of networked files are summarized (as returned in *st_mode* by *stat*(2)), but not copied to the new file.

AUTHOR

makecdf was developed by HP.

SEE ALSO

cdf(4), context(5), clusterconf(4), acl(5).

NAME

makedbm - make a Network Information System database

SYNOPSIS

/usr/etc/yp/makedbm [**-i** *nis_input_file*] [**-o** *nis_output_name*] [**-d** *nis_domain_name*] [**-m** *nis_master_name*] *infile outfile*
/usr/etc/yp/makedbm -u *database_name*

DESCRIPTION

makedbm generates databases (maps) for the Network Information System (NIS) from *infile*. A database created by **makedbm** consists of two files: *outfile*.*pag* and *outfile*.*dir*. A **makedbm** database contains records called **dbm records** composed of key-value pairs.

Each line of *infile* is converted to a single dbm record; all characters up to the first tab or space form the key, and the remainder of the line is the value. If a value read from *infile* ends with \, the value for that record is continued onto the next line. The NIS clients must interpret the # character (which means that **makedbm** does not treat the # as if it precedes a comment). If *infile* is a hyphen (-), **makedbm** reads standard input.

makedbm always generates a special dbm record with the key **YP_LAST_MODIFIED**, whose value is the time of last modification of *infile* (or the current time, if *infile* is -). This value is also known as the order number of a map, and **yppoll** prints it for a specified NIS map (see *yppoll*(1M)).

Another special dbm record always created by **makedbm** has the key **YP_MASTER_NAME**. Its value is usually the host name retrieved by **gethostname()**; however, the **-m** option can be used to specify a different value (see *gethostname*(2)).

Options

makedbm recognizes the following options and command-line arguments.

-i Create a special dbm record with the key **YP_INPUT_FILE** and the value **nis_input_file**.

-o Create a special dbm record with the key **YP_OUTPUT_NAME** and the value **nis_output_name**.

-d Create a special dbm record with the key **YP_DOMAIN_NAME** and the value **nis_domain_name**.

-m Replace the value of the special dbm record whose key is **YP_MASTER_NAME** with **nis_master_name**.

-u Undo the *database_name* (i.e., write the contents of *database_name* to the standard output), one dbm record per line. A single space separates each key from its value.

EXAMPLES

Shell scripts can be written to convert ASCII files such as **/etc/netgroup** to the key-value form used by **makedbm**. For example,

```
#!/bin/sh
/bin/awk 'BEGIN { FS = ":" } { print $1, $0 }' \
      /etc/netgroup | \
makedbm - netgroup
```

converts the file **/etc/netgroup** to a form that is read by **makedbm** to make the NIS map **netgroup**. The keys in the database are *netgroup*(4) names, and the values are the remainders of the lines in the **/etc/netgroup** file.

AUTHOR

makedbm was developed by Sun Microsystems, Inc.

SEE ALSO

domainname(1), ypinit(1M), ypmake(1M), yppoll(1M), gethostname(2), netgroup(4), ypfiles(4).

INTERNATIONAL SUPPORT

8-bit data, 8-bit file names, messages

NAME

mirror - disk mirroring utility

SYNOPSIS

mirror -c [**-f**] *primarydev pstate secondarydev sstate*

mirror -u *mirrordev ...*

mirror -o [**-f**] **-p** | **-s** *mirrordev* [[**-p** | **-s**] *mirrordev ...*]

mirror -r [**-t**] *mirrordev*

mirror -l [*device ...*]

Remarks:

This command requires installation of optional DataPair/800 software (not included in the standard HP-UX operating system) before it can be used.

DESCRIPTION

mirror allows the System Administrator to configure, unconfigure, and control *mirrored disks*. A mirrored disk is a pair of disk sections that contain the same data, and that look to the user like a single section. The mirror driver automatically maintains the copies by sending writes to both sections, and sending reads to either one.

mirror must be used with exactly one of the options **-c**, **-u**, **-o**, **-r**, or **-l**.

Configure a Mirror

mirror -c [**-f**] *primarydev pstate secondarydev sstate*

Configure a mirror. *Primarydev* and *secondarydev*, the *primary device* and the *secondary device*, are the names of the two sections in the mirror. These must be the same sections on different drives of the same model of disk. The devices cannot already be in a mirror.

The device names can be either block or character devices. The result is the same in either case: both the block and the character interface will be mirrored.

Once the mirror is configured, the block and character names of the primary device become the names of the mirror itself. The mirror can be read, written, or mounted, just like any other disk section.

pstate and *sstate* are the initial states of the primary and secondary. The states can be **online** or **offline**. One of the two devices must be ONLINE. The other can be ONLINE or OFFLINE. ONLINE sections are written and read by the mirror driver as required. OFFLINE sections are normally not accessed.

Warning: Do not set both sections in a mirror ONLINE unless both sections have exactly the same data, or unless the current data will not be read. If both sections are not exactly the same, set one section OFFLINE, then use **mirror -r** (reimage).

The primary device may be in use (open or mounted), but if it is, it must be declared ONLINE, and the secondary must be declared OFFLINE. The secondary device cannot be in use.

-f manually sets the *fail flag* for the OFFLINE section. This is a flag displayed by **mirror -l** (list), but not otherwise interpreted by the mirror driver. It is meant to indicate that the device is OFFLINE because of a hardware failure.

It is not possible to configure root and swap mirrors with **mirror -c** unless these mirrors have been put into the kernel by *uxgen*(1M).

Unconfigure a Mirror

mirror -u *mirrordev ...*

Unconfigure the named mirror(s). The two sections composing the mirror revert to their original, unmirrored behavior.

The mirror may be in use, but if it is, the primary device must be ONLINE. If the mirror is in use, the primary device is in use after **mirror -u** completes.

It is not possible to unconfigure a mirror if one section is in the REIMAGE state, caused by a running **mirror -r**. To unconfigure a mirror being reimaged, kill the **mirror -r** first.

Take One Section of a Mirror
mirror -o [**-f**] **-p** | **-s** *mirrordev* [[**-p** | **-s**] *mirrordev* ...]

Take one section of *mirrordev* OFFLINE. **-p** requests that the primary go OFFLINE, and **-s** the secondary. **-f** causes the fail flag to be set.

Typically the System Administrator sets a section OFFLINE so that the mirror can be backed up or repaired. The section is brought ONLINE again with **mirror -r** (reimage). Applications using the mirror do not notice these state changes.

One section in a mirror must always be ONLINE. Thus, it is not possible to take the primary OFFLINE unless the secondary is ONLINE
and vice versa.

If the mirror driver detects a device failure, it changes that section's state to OFFLINE automatically, setting the fail flag, as long as the other section is ONLINE. If the other section is not ONLINE, the mirror does not change state and the failure is handled as it would be for an unmirrored device.

Sometimes sections in several mirrors must go OFFLINE simultaneously, for example because a single data base running on several mirrors needs to be backed up. To do this, list the sections on a single command line, interspersing **-p** and **-s** as necessary.

To backup or repair a section once it is OFFLINE, whether it is the primary or the secondary, use the secondary name, which is the *offline-access device* while the mirror is configured. As with the mirror, both block and character interfaces can be used. It is not possible to write to the offline-access device. It is not possible to open the offline-access device unless one section is OFFLINE. It is not possible to change state while the offline-access device is open.

This command can also be used to set or clear the fail flag for a single mirror, even when the device is already OFFLINE.

Reimage a Mirror
mirror -r [**-t**] *mirrordev*

Reimage the named mirror. This happens in three steps. First, the OFFLINE section's state changes to REIMAGE. Second, the mirror driver copies data from the ONLINE section to the REIMAGE section until the two copies are identical. Third, the section's state changes from REIMAGE to ONLINE. Only then does the command complete.

One section must be OFFLINE. The offline-access device must not be open.

-t requests a *table-driven* reimage. When a section goes OFFLINE, either through **mirror -o** or because of device failure, the driver starts keeping a table that records subsequent writes to the ONLINE side. A table-driven reimage restores only those blocks. The table is in memory, and is lost across reboots.

Without **-t**, a *full* reimage is performed, which copies every block in the section. A full reimage is required if the disk was replaced while the section was OFFLINE. A table-driven reimage is an error in this case, but this error is undetectable by *mirror*. *mirror* requires a full reimage if the system has rebooted since the section went OFFLINE.

List Mirrors
mirror -l [*device* ...]

List mirrors. With no devices named, all configured mirrors are listed. Otherwise, mirrors containing the given devices are listed.

Output is formatted, one line per mirror, as follows:

primarydev pstate secondarydev sstate fail

primarydev and *pstate* are the primary device and its state. The state is given as **ONLINE**, **OFFLINE**, or **REIMAGE**. Similarly, *sstate* is the state of *secondarydev*. *fail* is **FAIL** if the fail flag is set, and **GOOD** otherwise.

If a *device* is specified, and it is not mirrored, its state is shown as **UNCONF**, and no secondary or failure information is printed.

If the **-m**, **-mp**, or **-ms** options to *hpuxboot*(1M) have been used to select which side of the root mirror to put ONLINE, and the side selected is the one that would normally be OFFLINE, the two states are shown as **ONLINE** and **NOREIM**. This indicates to *mirrorrc*(1M) that *mirrortab*(1M) is not reliable, and that no mirrors should be automatically reimaged.

Safety Protections

mirror requests other than **-l** are rejected if the *mirrorlog* daemon is no longer alive. Also, the daemon's death can cause some mirrored writes to block indefinitely. These are safety precautions that avoid possible loss of data. To recover, restart *mirrorlog*.

mirror fails when disk mirroring is not configured into the kernel, or when the request is issued on a client cnode in an HP-UX cluster.

EXAMPLES

Mirror **/extra**, currently mounted on **/dev/dsk/c2000d0s5**, with **/dev/dsk/c2001d0s5**, which is free:

$ mirror -c /dev/dsk/c2000d0s5 online /dev/dsk/c2001d0s5 offline

Reimage the OFFLINE section (full reimage):

$ mirror -r /dev/dsk/c2000d0s5

Set the primary OFFLINE for backup:

$ mirror -o -p /dev/dsk/c2000d0s5

After the backup, which reads from **/dev/[r]dsk/c2001d0s5**, reimage the OFFLINE section using the table:

$ mirror -r -t /dev/dsk/c2000d0s5

Describe all mirrors:

$ mirror -l
/dev/dsk/c2000d0s5 ONLINE /dev/dsk/c2001d0s5 ONLINE GOOD

HARDWARE DEPENDENCIES

Requires HP 9000 Series 800 with HP 7936FL/7937FL disks.

SEE ALSO

brc(1M), hpuxboot(1M), mirrorlog(1M), uxgen(1M), mirrortab(4).

NAME

/etc/mirrorlog - state-change logger for mirror disk subsystem

SYNOPSIS

mirrorlog [**-p** *progname*]

Remarks:

This command requires installation of optional DataPair/800 software (not included in the standard HP-UX operating system) before it can be used.

DESCRIPTION

In response to state changes in any disk mirror, **mirrorlog** writes a description of all mirrors to log file, **/etc/mirrortab**. This file is used to reconfigure mirrors after a reboot. See *mirrortab*(4) for a description of the format.

The **-p** option causes **mirrorlog** to execute *progname* whenever a mirror's fail flag comes on. This allows the System Administrator to take special action, such as mailing notifications or logging, in the case of disk failures.

mirrorlog can be executed only by users with appropriate privileges.

One and only one **mirrorlog** process should be running at all times. If **mirrorlog** dies, mirroring continues to function, except that **mirror** requests other than **-l** are rejected, and some mirrored writes block indefinitely. Diagnostic messages appear on the system console. To recover, restart **mirrorlog**.

AUTHOR

mirrorlog was developed by HP.

FILES

/dev/rdsk/mirconfig access to mirror driver

/etc/mirrortab log file

SEE ALSO

mirror(1m), mirrortab(4).

NAME

mkboot, rmboot - install, update, or remove boot programs from a disk device

SYNOPSIS

/etc/mkboot [**-b** *boot_file_path*] [**-c**] [**-f**] [**-h**] [**-s** *series*] [**-u**] [**-v**] *device*
/etc/mkboot [**-b** *boot_file_path*] [**-i** *included_lif_file*] [**-l**] [**-p** *preserved_lif_file*] [**-s** *series*] [**-v**] *device*
/etc/mkboot [**-a** *auto_file_string*] [**-s** *series*] [**-v**] *device*
/etc/rmboot *device*

DESCRIPTION

mkboot is used to install or update boot programs on the specified device file.

Options

mkboot recognizes the following options:

-a *auto_file_string*
This option is valid only for series 800 boot programs. If the **-a** option is specified, **mkboot** creates an *autoexecute* file **AUTO** on *device* if none exists. **mkboot** deposits *auto_file_string* in that file. If this string contains spaces, it must be quoted so that it is a single parameter.

-b *boot_file_path* If this option is given, boot programs in the pathname specified by *boot_file_path* are installed on the given device.

-c This option is valid only for series 700 boot programs. If this option is specified, **mkboot** checks the available space on the *device*. If the boot programs can fit in the available space, **mkboot** exits with a zero status. If the boot programs are too large to fit in the available space, **mkboot** exits with a status code of 1. If the verbose option is selected along with this option, a message is also displayed to the standard output.

-f This option is valid only for series 700 boot programs. This option should only be used when the system is in the single user state. Specifying this option modifies the LIF contents to reflect the information contained in the *boot_file_path* on the named *device*. This option is provided as a means for forcing the information contained in *boot_file_path* to be placed on the specified device without regard to the current swapping status. Its intended use is to allow the boot area to grow without having to boot the system twice (see the **-h** option). This could be a dangerous operation because swap space that is already allocated and possibly in use will be overwritten by the new boot program information. A message is also displayed to the standard output stating that the operator should immediately reboot the system to avoid system corruption and to reflect new information on the running system.

-h This option is valid only for series 700 boot programs. Specifying this option shrinks the available space allocated to swap in the LIF header by the amount required to allow the installation of the new boot programs specified by *boot_file_path*. After the LIF header has been modified, reboot the system to reflect the new swap space on the running system. At this point, the new boot programs can be installed and the system rebooted again to reflect the new boot programs on the running system. This is the safe method for accomplishing the capability of the **-f** option.

-i *included_lif_file*
This option is valid only for series 800 boot programs. If the **-i** option is specified one or more times, **mkboot** copies each *included_lif_file* specified and ignores any other LIF files in *boot_file_path*. The sole exceptions to this rule are the files **ISL** and **HPUX**, which are copied without regard to the **-i** options. If *included_lif_file* is also specified with the **-p** option, the **-i** option is ignored. If the **-i** option is used with **LABEL** as its argument and the file **LABEL** does not exist on *boot_file_path*, then if *device* is an LVM physical volume or the **-l** option is used, **mkboot** creates a minimal **LABEL** file on *device* which will permit the system to boot on *device*, possibly without swap or dump.

`-l` This option is valid only for series 800 boot programs. If this option is used, `mkboot` treats *device* as an LVM physical volume, regardless of whether or not it is currently set up as one.

`-p` *preserved_lif_file*
This option is valid only for series 800 boot programs. If the `-p` option is specified one or more times, `mkboot` keeps each specified *preserved_lif_file* intact on *device*. If *preserved_lif_file* also appears as an argument to the `-i` option, that `-i` option is ignored. This option is typically used with the *autoexecute* file `AUTO` and with the LVM and SwitchOver/UX file `LABEL`. If *preserved_lif_file* is not on *device*, `mkboot` fails.

`-s` *series* If this option is specified, boot programs for the given series are installed on the given device. *series* must be `300`, `700`, or `800`. Note that in this context, Series 400 is treated as being identical to Series 300. If `700` is specified, `mkboot` by default installs boot programs from file `/usr/lib/uxbootlf.700`. If `800` is specified, `mkboot` by default installs boot programs from file `/usr/lib/uxbootlf`. If `300` is specified, `mkboot` by default installs boot programs from `/etc/boot`. These defaults are overridden by the `-b` option. If the `-s` option is not specified, `mkboot` defaults to the series of the current machine.

`-u` This option is valid only for Series 700 boot programs. If `-u` is specified, `mkboot` uses the information contained in the LIF header to identify the location of swap area, boot area, and raw I/O so that installation of the boot programs does not violate any user data. Normally, the LIF header information is overwritten on each invocation of `mkboot`. This option is provided to allow modification of boot programs on a Series 700 disk that is also actively supporting swap and/or raw I/O.

`-v` This option is meaningful only for Series 700 boot programs. If this option is specified, `mkboot` displays its actions including the amount of swap space available on the specified device.

device Install the boot programs on the given device special file. The specified device can identify either a character-special or block-special device. However, because of the operations performed on the specified device file, `mkboot` requires that both be present. `mkboot` attempts to determine whether a device is character- or block-special by examining the corresponding specified path name. For this reason, the complete path name must be supplied. If `mkboot` is unable to determine the corresponding device file, a message is written to the display, and `mkboot` exits.

`rmboot` removes the boot programs from the boot area.

AUTHOR

`mkboot` and `rmboot` were developed by HP.

WARNINGS

On Series 700 systems, in order for `mkboot` to determine the layout of the disk, a file system must reside on the device being modified.

Since the boot area is taken from swap space on the series 700, `mkboot` cannot increase the amount of space allocated to boot programs on a disk where swap and/or raw I/O are currently enabled.

When executing from a recovery system, the `mkboot` command (if used) must be invoked with the `-f` option; otherwise it will not be able to replace the boot area on your disk.

If *device* is, or is intended to become an LVM physical volume on a Series 800 system, *device* must specify section 2.

DEPENDENCIES

Series 300/400:

The `-a`, `-c`, `-f`, `-h`, `-i`, `-l`, `-p`, and `-u` options are not supported.

Series 700:

The `-a`, `-i`, `-l`, and `-p` options are not supported.

Series 800:

The `-c`, `-f`, `-h`, and `-u` options are not supported. Boot programs are stored in the boot area in Logical Interchange Format (LIF), which is similar to a file system. In order for a device to be bootable, the LIF volume on that device must contain at least the **ISL** (the initial system loader) and **HPUX** (the HP-UX bootstrap utility) LIF files. If, in addition, the device is an LVM physical volume, the **LABEL** file must be present (see *lvlnboot*(1M)).

FILES

`/usr/lib/uxbootlf`	file containing series 800 boot programs
`/usr/lib/uxbootlf.700`	file containing series 700 boot programs
`/etc/boot`	file containing series 300 boot programs
ISL	initial system loader
HPUX	HP-UX bootstrap and installation utility
AUTO	defines default/automatic boot behavior (see *hpux*(1M))
LABEL	used by SwitchOver/UX and LVM
RDB	diagnostics tool
IOMAP	diagnostics tool

SEE ALSO

boot(1M), hpux(1M), isl(1M), lif(4), lvlnboot(1M), mkfs(1M), newfs(1M).

NAME

mkdev - make device files

SYNOPSIS

`/etc/mkdev`

DESCRIPTION

mkdev is a shell script to help the system administrator install and maintain an HP-UX system. It consists of a machine-dependent list of commands that create one of each possible type of device file, with suggested default device addresses. It also creates mount directories for mountable volumes and changes permissions as appropriate for the device files.

mkdev makes it easier to build (or rebuild) special files in a single operation.

mkdev automatically changes the working directory (using the **cd** command) to **/dev** before starting execution.

mkdev is specifically intended for modification before (each) use. Command lines for non-desired devices should be commented out with **#** so that they are still available for later use. Shorter device names than those suggested may be preferred, especially for default devices. For HP-UX naming conventions, see *intro*(7).

SEE ALSO

chmod(1), mkdir(1), mknod(1M), intro(7).

DIAGNOSTICS

Each command line in **mkdev** is echoed as it is executed. Error messages, if any, are generated by the commands invoked.

To ensure that the file is modified before being used, an error is given if it has not been modified.

AUTHOR

mkdev was developed by HP.

NAME

mkfs - construct a file system

SYNOPSIS

/etc/mkfs [**-L**|**-S**] [**-F**] *special size* [*nsect ntrack blksize fragsize ncpg minfree rps nbpi*]
/etc/mkfs [**-L**|**-S**] [**-F**] *special proto* [*nsect ntrack blksize fragsize ncpg minfree rps nbpi*]

Remarks

HFS file systems are normally created with the **newfs** command (see *newfs*(1M)).

DESCRIPTION

mkfs constructs a file system by writing on the special file *special*. **mkfs** builds a file system with a root directory and a **lost+found** directory (see *fsck*(1M)). The **FS_CLEAN** magic number for the file system is stored in the super block.

mkfs creates the file system with a rotational delay value (see *tunefs*(1M)) as based on the interface (SCSI, HP-FL, HPIB) and other characteristics of the disk drive. To get the appropriate rotational delay value, turn immediate reporting on or off before rather than after creating the file system.

Options

mkfs recognizes the following options:

-L Build a long-filename file system that allows directory entries (file names) to be up to **MAXNAMLEN** (255) bytes long.

-S Build a short-filename file system that allows directory entries (file names) to be up to **DIRSIZ** (14) bytes long.

There are two types of HFS file systems, distinguished mainly by differing directory formats that place these different limits on the length of file names. If neither **-L** nor **-S** is specified, **mkfs** creates a file system of the same type as the root file system.

-F Force **mkfs** to process on a mounted file system.

This options is provided primarily for backward compatibility. If **-F** is specified, **mkfs** continues to process on a mounted file system. Otherwise, it prompts the user and waits for a response. However, **mkfs** does not work on a swap device, even if **-F** is specified.

Arguments

One of the following arguments is required after *special*:

size Specifies the number of **DEV_BSIZE** blocks in the file system (**DEV_BSIZE** is defined in **<sys/param.h>**.)

proto If *proto* is a file name that can be opened, **mkfs** assumes it to be a prototype file and takes its directions from that file. Prototype structure is described in detail below.

The following optional arguments allow fine-tune control over file system parameters:

nsect Number of sectors per track on the disk.

ntrack Number of tracks per cylinder on the disk.

blksize Primary block size for files on the file system. Must be a power of two. See DEPENDENCIES for allowable values.

fragsize Fragment size for files on the file system. *fragsize* represents the smallest amount of disk space to be allocated to a file. It must be a power of two not smaller than **DEV_BSIZE** and no smaller than one-eighth of the file system block size.

ncpg Number of disk cylinders per cylinder group. This number must be in the range 1 to 32.

minfree Minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only users with appropriate privileges can allocate disk blocks. The default value is 10%.

rps Number of disk revolutions per second. The default value is 60.

nbpi Number of data bytes (amount of user file space) per inode slot. The number of inodes is calculated as a function of the file system size. If *nbpi* is not valid, the default value of

2048 is used.

If the second argument is a file name that can be opened, **mkfs** assumes it to be a prototype file *proto*, and takes its directions from that file. The prototype file contains tokens separated by spaces or new-line characters.

Prototype File Structure

The prototype file contains tokens separated by spaces or new-line characters. The first token is the name of a file to be copied onto block zero as the bootstrap program (usually **/etc/BOOT**). If the name of a file is "", it is ignored. The second token is a number specifying the number of **DEV_BSIZE**-byte blocks in the file system. The next tokens comprise the specification for the root directory. File specifications consist of tokens giving the mode, user ID, group ID, and initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **-bcdlL** specify regular files, block special files, character special files, directories, symbolic links, and hard links respectively.) The second character of the type 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 three-digit octal number giving the *owner*, *group*, and *other* read, write, execute permissions (see *chmod*(1)).

Two tokens come after the mode that specify the user and group IDs of the owner of the file. These values can be specified numerically or by using symbolic names that appear in the password and group databases.

If the file is a regular file, the next token is a pathname from which the contents and size are copied.

If the file is a block or character special file, two numeric tokens follow which give the major and minor device numbers.

If the file is a directory, **mkfs** makes the entries **.** and **..**, then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

If the file is a symbolic link, the next token is a pathname that is used as the path to which the symbolic link should point.

If the file is a hard link, the next token is a pathname that is used as the name of the file within the new filesystem to which this entry should be linked. The mode, owner and group fields of this entry are ignored; they are taken from the target of the link. The target of the link must be listed before the entry specifying the link. Hard links to directories are not permitted.

With the exception of the mode field (which is always an octal number), all numeric fields can be specified in hexadecimal, octal or decimal using standard notation (values starting with a leading **0x** are hex; a leading **0** are octal; all others are decimal).

Here is a sample prototype specification:

```
/etc/BOOT
4872
d--555 bin  bin
bin     d--555 bin  bin
        sh      ---555 bin  bin /bin/sh
        rsh     L--555 bin  bin /bin/sh
        su      -u-555 root bin /bin/su
        $
dev     d--555 bin  bin
        b0      b--640 root sys 0 0x0e0000
        c0      c--640 root sys 4 0x0e0000
        $
etc     d--555 bin  bin
        init    ---555 bin  bin /etc/init
        passwd  ---444 bin  bin /etc/passwd
        yp      l--444 bin  bin /usr/etc/yp
        $
usr     d--555 bin  bin
        bin     d--555 bin  bin
                more    ---555 bin  bin /usr/bin/more
```

```
                    $
          $
$
```

Access Control Lists (ACLs)
Every file with one or more optional ACL entries consumes an extra (continuation) inode. If you anticipate significant use of ACLs on a new file system, you can allocate more inodes by reducing the value of *nbpi* appropriately. The small default value typically causes allocation of many more inodes than are actually necessary, even with ACLs. To evaluate your need for extra inodes, run `bdf -i` on existing file systems. For more information on access control lists, see *acl*(5).

DEPENDENCIES

Series 300/400/800:
Valid file system block sizes are 4096 and 8192.

Series 700:
Valid file system block sizes are 4096 and 8192 without Software Disk Striping, or 4096, 8192, 16 384, 32 768, and 65 536 with Software Disk Striping.

AUTHOR
`mkfs` was developed by HP and the University of California, Berkeley.

SEE ALSO
chmod(1), bdf(1M), fsck(1M), fsclean(1M), newfs(1M). dir(4), fs(4), group(4), passwd(4), symlink(4), acl(5).

NAME

mklost+found - make a lost+found directory for *fsck*(1M)

SYNOPSIS

`/etc/mklost+found`

DESCRIPTION

`mklost+found` creates a directory named **`lost+found`** in the current directory. It also creates several empty files which are then removed to provide empty slots for **`fsck`** (see *fsck*(1M)).

For the HFS file system, **`mklost+found`** is not normally needed, since **`mkfs`** automatically creates the **`lost+found`** directory when a new file system is created (see *mkfs*(1M)).

AUTHOR

`mklost+found` was developed by the University of California, Berkeley.

SEE ALSO

fsck(1M), mkfs(1M).

NAME

mknod - create special files

SYNOPSIS

/etc/mknod *name* **c** *major minor* [*cnode_name*]

/etc/mknod *name* **b** *major minor* [*cnode_name*]

/etc/mknod *name* **p**

DESCRIPTION

mknod creates the following types of files:

- Character device special file (first SYNOPSIS form),
- Block device special file (second SYNOPSIS form),
- FIFO file, sometimes called a named pipe (third SYNOPSIS form).

name is the path name of the file to be created. The newly created file has a default mode readable and writable by all users (066), but the mode is modified by the current setting of the user's file mode creation mask (see *umask*(1)).

Character and Block Special Files

Character device special files are used for devices that can transfer single bytes at a time, such as nine-track magnetic tape drives, printers, plotters, disk drives operating in "raw" mode, and terminals. To create a character special file, use **c** as the second argument to the **mknod** command.

Block device special files are used for devices that usually transfer a block of data at a time, such as disk drives. To create a block device special file, use **b** as the second argument to **mknod**.

The remaining arguments specify the device that will be accessible through the new special file:

major	This "major number" specifies the major device type (for example, the device driver number)
minor	This "minor number" specifies the device location (typically, but not always, the unit, drive, HP-IB bus address and/or line number).
cnode_name	If present, *cnode_name* specifies the cnode name, or if it is numeric, the cnode ID (see *glossary*(9)) from which the device special file can be accessed. For non-numeric cnode names, the file **/etc/clusterconf** is searched to determine the corresponding cnode ID.

The *major* and *minor* values can each be specified in hexadecimal, octal, or decimal, using C language conventions (decimal: no leading zero, octal: leading zero, hexadecimal: leading **0x**).

Assignment of major and minor device numbers is specific to each HP-UX system. Refer to the System Administrator manuals supplied with your system for details.

Only users who have appropriate privileges can use **mknod** to create a character or block device special file.

FIFO files

To create a FIFO (named pipe or buffer) file, use **p** as the second argument to **mknod** (the **mkfifo** command can also be used for this purpose — see *mkfifo*(1)). All users can use **mknod** to create FIFO files.

Access Control Lists (ACLs)

Optional ACL entries can be added to special files and FIFOs with *chacl*(1). However, system programs are likely to silently change or eliminate the optional ACL entries for these files.

WARNINGS

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP MACS software has been installed, access control lists are handled differently. Refer to HP MACS documentation for information about access control lists in the HP MACS environment.

FILES

/etc/clusterconf	For translating cnode names to cnode IDs

SEE ALSO

chacl(1), mkdir(1), mkfifo(1), umask(1), lsdev(1M), mkdev(1M), sam(1M), mknod(2), acl(5), mknod(5).

HP-UX System Administrator manuals.

STANDARDS CONFORMANCE

mknod: SVID2, XPG2

NAME

mkpdf - create a Product Description File from a prototype PDF

SYNOPSIS

mkpdf [**-c** *comment_string*] [**-n**] [**-r** *alternate_root*] *prototype_PDF new_PDF*

DESCRIPTION

mkpdf is a program that reads a prototype PDF and generates a new PDF (see *pdf*(4)) that reflects the current status of the file system files defined by pathnames in the prototype file.

If *pathname* is a directory, the *size*, *version*, *checksum*, and *linked_to* target fields are forced to be empty. If the file is a device, the *version*, *checksum*, and *linked_to* fields are forced to be empty and the *size* field contains the major and minor device numbers.

If a pathname in *prototype_PDF* is prefaced with "**?**", the file is assumed to be an optional file. This file is processed in the same manner as all other files except that if the file does not exist, values provided in the prototype are reproduced, and the "**?**", is passed through to *new_PDF*. If a pathname is not preceded with "**?**", and the file does not exist on the file system, an error is reported and no entry is added to *new_PDF*.

If - is used for either the *prototype_PDF* or *new_PDF*, *mkpdf* assumes that standard input and/or standard output is being used for the approprate value.

Comments in *prototype_PDF* are supported as follows: Lines beginning with the character '%' are generally passed through, in order, to *new_PDF*, except that any "**% Product Description File**" and "**% total size is ...**" lines are removed to prevent duplication of these automatically generated lines in *new_PDF* when *prototype_PDF* is a PDF. Lines beginning with the hash character **#**, and lines containing only the new-line character **are not passed through to** *new_PDF*. Note that blank space preceding these special characters is not allowed and will generally result in error messages about files not found.

A size summary is produced as a comment at the end of the PDF.

Options

-c *comment_string* A string that contains a comment about the product for which this PDF is being generated. This is used as a second comment line of the PDF. See *pdf*(4) for a description of the first comment line. If this option is not specified, no second comment line is produced.

-n Record numerical representation of user id from */etc/passwd* and group id from */etc/group* for each file instead of the usual text representation.

-r *alternate_root* *alternate_root* is a string that is prefixed to each pathname in the prototype (after removal of the optional **?**) to form a modified pathname to be used to gather attributes for the entry. Default is an empty string.

EXAMPLES

Given a file "Proto" with contents:

```
/bin/basename
/bin/cat
/bin/cc
/bin/dirname
/bin/grep
/bin/ls
/bin/ll::::::::/bin/ls
/bin/su
```

The command:

mkpdf -c "fileset TEST, Release 1.0" Proto -

produces the PDF shown in the EXAMPLE section of *pdf*(4).

The following example creates a totally new PDF for the fileset UX_CORE. The *pathname* and *linked_to* are taken from the prototype PDF. All other fields are generated from the file system.

mkpdf /tmp/UX_CORE /system/UX_CORE/new.pdf

The next example shows how to create a completely new *PDF* from just a list of files. The PDF for the files under the /PRODUCT directory is created by executing *find*(1) on all the files in the directory structure under */PRODUCT*. A / is edited onto the beginning of each pathname to make it absolute. The pathnames are then piped to *mkpdf*. A **-r** option specifies that a root of */PRODUCT* should be prefixed to each pathname while the directory is being searched. A - in the prototype PDF position specifies that *stdin* is being used for the prototype PDF file. The resultant PDF does not contain the /PRODUCT prefix. Note that with only a list of pathnames, the *linked_to* field of linked files will not conform to the convention explained in *pdf*(4).

```
cd /PRODUCT
find * -print | sed -e 's:^:/:' |
mkpdf -r /PRODUCT - PDF
```

RETURN VALUE

Upon completion, *mkpdf* returns one of the following values:

1 Returned if non-optional files in the prototype file were not found.

2 Returned if *mkpdf* encountered other problems.

0 Successful completion.

DIAGNOSTICS

filename: no such file or directory

A non-optional file was not found on the file system and will not appear in the new PDF.

WARNINGS

Sizes reported do not reflect blocks allocated to directories (including CDFs).

AUTHOR

mkpdf was developed by HP.

SEE ALSO

pdfck(1M), pdfdiff(1M), pdf(4).

NAME

mkrs - construct a recovery system

SYNOPSIS

/etc/mkrs [**-v**][**-q**][**-s**][**-f** *rcdev*][**-r** *rootdev*][**-m** *series*]

DESCRIPTION

mkrs constructs a recovery system on removable media (or a hard disk drive). If a system is unbootable due to a corrupt root disk, the system administrator boots the recovery system and uses it as a temporary root volume. Once booted on the recovery system, the administrator uses the tools it provides to repair the corrupt root disk.

Options

mkrs recognizes the following *options*:

-q If enough free disk space is available in **/usr/tmp** (typically 10-20Mb), the **-q** option can be used to make **mkrs** create an image of the recovery system in this directory before copying it to the recovery media. This option generally saves a great deal of time due to reduced seeking on non-random-access recovery media (cartridge tape and DDS). Note: for DDS tape recovery systems, the **-q** option is assumed.

-s When creating a DDS recovery system for a small memory system (8Mb or less), the **-s** option should be used to specify that a smaller set of files be placed on the recovery system.

-v (verbose) Prints a running history of the construction process. Normally, **mkrs** does its work silently.

-f *rcdev* Specify the name of the device file for the recovery system (that is, the cartridge tape drive, magneto-optical disk drive, or alternate hard disk drive on which the recovery system is created). **mkrs,** by default, uses the following device file:

/dev/update.src	if it exists as a character device file, else
/dev/rct/c0	if it exists as a character device file, else
/dev/rct	if it exists as a character device file, else the device file must be specified.

If none of the above defaults exist on the system, one of these device files must be created or the **-f** option must be used to specify the device file to be used (the recovery device file can be either a block or a character device file).

-r *rootdev*

Specify the name of the device file for the root device. **mkrs**, by default, uses the following device file:

/dev/dsk/0s0	if it exists as a block device file, else
/dev/root	if it exists as a block device file, else
/dev/hd	if it exists as a block device file, else the device file must be specified.

If none of the above defaults exist on the system, one of these device files must be created or the **-r** option must be used to specify the device file to be used (the root device file must be a block device file).

-m *series*

Specify which type of machine is running this software (for example, **-m 300**). Normally, **mkrs** properly identifies the machine type. This option can be used if **mkrs** is unable to identify the machine type.

DIAGNOSTICS

An error message results if:

- None of the default device files for the recovery device exist and the **-f** option is not used to specify a recovery device file.
- None of the default device files for the root device exist and the **-r** option is not used to specify a root device file.
- The machine type cannot be determined and the **-m** option is not used to specify the machine type.

WARNINGS

Incorrectly specifying the recovery device may cause file system damage during recovery system construction.

The recovery system provides super-user capabilities; the system administrator should have exclusive responsibility for its use.

The recovery system uses the swap area of the system being repaired for its swap space.

Recovery systems that are created on DDS tape devices will only work if the DDS tape device has a SCSI interface.

mkrs expects that the recovery media specified has already been formatted with **mediainit** in the case of magneto-optical or hard disk drives.

When executing from a recovery system, the **mkboot** command (if used) must be invoked with the **-f** option; otherwise it will not be able to replace the boot area on your disk.

DEPENDENCIES

Series 300

Series 300 systems must have Revision D or newer boot ROMs to support DDS tape recovery systems.

Series 700

The **-s** option is necessary for building Series 700 DDS tape recovery systems.

AUTHOR

mkrs was developed by HP.

SEE ALSO

HP-UX System Administrator manuals.

config(1M), mkfs(1M).

NAME

mksf - make a special file

SYNOPSIS

/etc/mksf [**-N** *cnode*] [**-d** *driver* | **-C** *class*] [**-H** *hw_path*] [**-l** *lu*] [*driver_options ...*] [*special_file*]

/etc/mksf [**-N** *cnode*] [**-d** *driver* | **-C** *class*] [**-H** *hw_path*] [**-r**] **-m** *minor special_file*

DESCRIPTION

mksf makes a special file for an existing device; that is, a device that has already been assigned a logical unit number by **insf** (see *insf*(1M)). The device is specified by supplying some combination of the **-d**, **-C**, **-H**, and **-l** options. If the options specified match a unique device in the system, **mksf** creates a special file for that device; otherwise, **mksf** prints an error message and exits.

For most drivers, **mksf** has a set of built-in driver options and special-file naming conventions. By supplying some subset of the driver options, as in the first form above, the user can create a special file with a particular set of characteristics. If a special-file name is specified, **mksf** creates the special file with that special file name; otherwise, the default naming convention for the driver is used.

In the second form, the minor number and special-file name are explicitly specified. This form is used to make a special file for a driver without using the built-in driver options in **mksf**. The **-r** option specifies that **mksf** should make a character (raw) device file instead of the default block device file for drivers that support both.

The **-N** option specifies that the special files are to be created with the associated *cnode* ID; the format of *cnode* is the same as that given in *mknod*(1M). If **-N** is not specified, **insf** uses the cnode ID of the machine on which it is executing.

Options

mksf recognizes the following options:

-C *class* Match a device that belongs to the specified device class. Device classes are defined in file **/etc/master**.

-H *hw_path* Match a device at the specified *hw_path*. *hw_path* specifies the address of the hardware components leading to a device. It consists of a string of numbers each suffixed by slash (**/**), followed by an arbitrary-length string of numbers separated by periods, (**.**). Hardware components suffixed by slashes indicate bus converters and may not be necessary on your machine. Hardware components suffixed by (**.**) indicate the addresses of the remaining hardware components on the path to a device.

-N *cnode* Make a special file with the specified *cnode* ID; the format of *cnode* is the same as that given in *mknod*(1M).

-d *driver* Match a device controlled by the specified *driver*.

-l *lu* Match a device with the specified logical unit number.

-m *minor* Create the special with the following *minor* number; the format of the *minor* number is the same as that given in *mknod*(1M).

-r Create a character (raw) special file instead of a block special file.

Driver-specific options and default special file names are listed below.

autox0/autoch

-z *cartridge* The cartridge number (start with 1).

-r Raw; create character, not block, special file.

-s *section* The section number.

special_file The default special file name depends on the whether the `-r` option is used.

-r	Special File Name
yes	`rac/cf`*Plu* `s` *section* `d` *cartridge* `a and` `rac/cf`*Plu* `s` *section* `d` *cartridge* `b`
no	`ac/cf`*Plu* `s` *section* `d` *cartridge* `a and` `ac/cf`*Plu* `s` *section* `d` *cartridge* `b`

`disc1`

`-c` This option must be present if the unit is a cartridge tape.

`-t` Transparent mode (normally used by diagnostics).

`-u` *unit* The CS/80 unit number (for example, unit 0: disk, unit 1: tape).

`-r` Raw; create character, not block, special file.

`-s` *section* The section number.

special_file The default special file name depends on whether the `-r` and `-c` options are used:

-r	-c	Special File Name
yes	yes	`rct/cf2lu d` *unit* `s` *section*
yes	no	`rdsk/cf2lu d` *unit* `s` *section*
no	yes	`ct/cf2lu d` *unit* `s` *section*
no	no	`dsk/cf2lu d` *unit* `s` *section*

`disc2`

`-t` Transparent mode (normally used by diagnostics).

`-u` *unit* The cs80 unit number (typically 0).

`-r` Raw; create character, not block, special file.

`-s` *section* The section number.

special_file The default special file name depends on whether the `-r` option is used:

-r	Special File Name
yes	`rdsk/cf`*Plu* `d` *unit* `s` *section*
no	`dsk/cf`*Plu* `d` *unit* `s` *section*

`disc3`

`-r` Raw; create character, not block, special file.

`-s` *section* The section number.

special_file The default special file name depends on whether the `-r` option is used:

-r	Special File Name
yes	`rdsk/cf`*Plu* `d0` *unit* `s` *section*
no	`dsk/cf`*Plu* `d0` *unit* `s` *section*

`display0`

`-D` *device* Specifies the device type. Possible values for *device* are `framebuf`, `hil`, `hilkbd`, and `ite`.

`-a` *address* The link address (1-7).

`-t` Transparent mode (normally used by diagnostics).

special_file The default special file name depends on the arguments `-D` and `-t`:

-D arg	-t	Special File Name
-D framebuf	no	crtf2lu
-D framebuf	yes	diag/crtf2lu
-D hil	no	hil_f2lu.address
-D hil	yes	diag/hilf2lu
-D hilkbd	no	hilkbdf2lu
-D ite	no	ttyif2lu

gpio0

-t Transparent mode (normally used by diagnostics).

special_file The default special file name is **gpio***lu*.

gpio1

-t Transparent mode (normally used by diagnostics).

special_file The default special file name is **gpio***lu*.

instr0

-a *address* The HP-IB instrument address (0-30).

-r Raw; the special file has no associated HP-IB instrument address.

-t Transparent mode (normally used by diagnostics).

special_file The default special file name is **hpib/***lu* **a***address* or **hpib/***lu* (if **-r**).

lpr0/lpr1/lpr2

-c Capital letters. Convert all output to uppercase.

-n No form-feed.

-r Raw.

-o Old paper-out behavior (abort job).

-e Eject page after paper-out recovery.

-t Transparent mode (normally used by diagnostics).

special_file The default special file name is **lp***lu* or **rlp***lu* (if **-r**).

mux0/mux0_16

-c CCITT.

-h Hardwired (direct connect).

-i Callin.

-o Callout.

-p *port* Multiplexer port number (0-5 for **mux0**; 0-15 for **mux0_16**).

-t Transparent mode (normally used by diagnostics).

special_file The default special file name is **tty***lu***p***port* or **mux***lu* (if **-t**).

cn/devconfig/diag0/dmem/meas_drivr/mm/pty0/pty1/sw/sy/clone/dlpidrv

-m *minor* The minor number.

special_file No default. Path must be specified.

tape1/tape2

-a AT&T style rewind/close.

-b *bpi* Bits per inch. Recognized values for *bpi* are: **800**, **1600**, **6250**.

`-c`	RTE compatible close.
`-n`	No rewind on close.
`-r`	Raw; create character, not block, special file.
`-t`	Transparent mode (normally used by diagnostics).
`-u`	UC Berkeley style rewind/close.
`-w`	Wait (disable immediate reporting).
`-C`	Enable data compression.
special_file	The default special file name is `mt`/*lu*`l`. The `-r` option changes `mt` to `rmt`. The `l` means low density (800 bpi). For 1600 and 6250 bpi, `l` is replaced by `m` (medium) and `h` (high), respectively. An `n` is appended to the name if the `-n` (no rewind) option is given. A `c` is appended to the name if the `-C` (compression) option is given.

RETURN VALUE

`mksf` returns `0` upon normal completion and `1` if an error occurred.

DIAGNOSTICS

Most of the diagnostic messages from `mksf` are self explanatory. Listed below are some messages deserving further clarification. Errors cause `mksf` to abort immediately.

Errors

`Ambiguous device specification`
Matched more than one device in the system. Use some combination of the `-d`, `-C`, `-H`, and `-l` options to specify a unique device.

`No such device in the system`
No device in the system matched the options specified. Use `ioscan` to list the devices in the system (see *ioscan*(1M)).

`Device driver` *name* `is not in the kernel`
`Device class` *name* `is not in the kernel`
The indicated device driver or device class is not present in the kernel. Add the appropriate device driver and/or device class to the `uxgen` input file and generate a new kernel (see *uxgen*(1M)).

`Device has no logical unit number`
The specified device has not been assigned a logical unit (*lu*) number. Use `insf` to assign an *lu* to the device.

EXAMPLES

Make a special file named `/dev/printer` and map it to the line printer device associated with logical unit number 2.

```
mksf -C printer -l 2 /dev/printer
```

Make a special file, using the default naming convention, for the tape device at hardware path 8.4.1. The driver-specific options specify raw mode, 1600 bits per inch, and no rewind on close.

```
mksf -C tape -H 8.4.1 -r -b 1600 -n
```

AUTHOR

`mksf` was developed by HP.

FILES

`/dev/config`
`/etc/master`

SEE ALSO

insf(1M), ioscan(1M), lssf(1M), mknod(1M), rmsf(1M), uxgen(1M), ioconfig(4).

NAME

mount, umount - mount and unmount file system

SYNOPSIS

/etc/mount [*fsname directory* [**-frv**] [**-s**|**-u**] [**-o** *options*] [**-t** *type*]]
/etc/mount -a [**-fv**] [**-s**|**-u**]
/etc/mount [**-p**] [**-l**|**-L**] [**-s**|**-u**]

/etc/umount [**-v**] [**-s**] *fsname*
/etc/umount [**-v**] [**-s**] *directory*
/etc/umount -a [**-v**] [**-s**] [**-h** *host*]] [**-t** *type*]]

DESCRIPTION

mount announces to the system that a removable file system is to be attached to the file tree at *directory*. *directory* must already exist, and becomes the name of the root of the newly mounted file system. *directory* must be given as an absolute path name and cannot be a context-dependent file (see *cdf*(4)). *fsname* must be either the name of a special file or of the form *host*:*path*. If *fsname* is of the form *host*:*path*, the file system type is assumed to be **nfs** (see **-t** option below).

These commands maintain a table of mounted devices in **/etc/mnttab**. If invoked with no arguments, **mount** prints the table.

umount announces to the system that the removable file system *fsname* previously mounted on directory *directory* is to be unmounted. Either the file system name or the directory where the file system is mounted can be specified.

In the HP Clustered Environment, only NFS and HFS file systems can be mounted and unmounted from client nodes. See Networking Features below.

Options (mount)

mount recognizes the following options. Options are position-independent and can occur in any order.

-a Attempt to mount all file systems described in **/etc/checklist**. All optional fields in **/etc/checklist** must be included and supported. If *type* is specified, all file systems in **/etc/checklist** with that *type* are mounted. File systems are not necessarily mounted in the order listed in **/etc/checklist**.

-f Force the file system to be mounted, even if the file system clean flag indicates that the file system should have **fsck** run on it before mounting (see *fsck*(1M)).

-p Print the list of mounted file systems in a format suitable for use in **/etc/checklist**.

-l In the HP Clustered environment, prints only HFS and CDFS file systems mounted on the local cnode (NFS mounts are not displayed).

-L In the HP Clustered environment, prints only file systems which may be unmounted from the local cnode. (Includes file systems mounted on the local node and cluster-wide NFS mounts).

-r Mount the specified file system as read-only. This option implies **-o ro**. Physically write-protected file systems must be mounted in this way or errors occur when access times are updated, whether or not any explicit write is attempted.

-u Force an update of **/etc/mnttab** from the kernel mount table.

-s Do not update the **/etc/mnttab** file with kernel mount information. Use of this option is provided for special cases of backward compatilibity only and is strongly discouraged. This option may be removed in a future release.

-t *type* Specify a file system *type*. Acceptable types are **hfs**, **cdfs**, and **nfs** (see *checklist*(4)). If **-a** is not used, the single file system specified is mounted as that *type*.

-v Verbose mode. Write a message to the standard output indicating which file system is being mounted.

-o *options* Specify a list of comma-separated *options* from the list below. Some *options* are valid for any file system *type*, while others apply only to a specific *type*.

The following *options* are valid on all file systems:

`defaults`	Use all default options. When used, this must be the only option specified.
`rw`	Read-write (default).
`ro`	Read-only.
`suid`	Set-user-ID execution allowed (default).
`nosuid`	Set-user-ID execution not allowed.

The following *options* are valid only on `hfs` type file systems:

`quota`	Disk quotas enabled (valid only for `rw` type file systems).
`noquota`	Disk quotas disabled (default).

Mounting with the *quota* option also *enables* quotas for the file system, unlike some other systems which require the additional invocation of the `quotaon` command after the file system has been mounted (see *quotaon*(1M)). Running `quotaon` does no harm, but is not necessary.

Options (umount)

`umount` recognizes the following options. Options are position-independent and can occur in any order.

`-a`	Unmount all locally mounted and NFS mounted file systems described in `/etc/mnttab`.
`-s`	Do not update the `/etc/mnttab` file with kernel mount information. Use of this option is provided for special cases of backward compatilibity only and is strongly discouraged. This option may be removed in a future release.
`-h` *host*	Unmount only those file systems listed in `/etc/mnttab` that are remote-mounted from *host*.
`-t` *type*	Unmount only file systems mounted with the given *type*.
`-v`	Verbose mode. Write a message to the standard output indicating which file system is being unmounted.

NETWORKING FEATURES

NFS

The following *options* are specific to `nfs` file systems:

`acregmin=`*n*	Hold cached attributes for at least *n seconds after file modification.*
`acregmax=`*n*	Hold cached attributes for no more than *n* seconds after file modification.
`acdirmin=`*n*	Hold cached attributes for at least *n* seconds after directory update.
`acdirmax=`*n*	Hold cached attributes for no more than *n* seconds after directory update.
`actimeo=`*n*	Set min and max times for regular files and directories to *n* seconds.
`bg`	If the first *mount* attempt fails, retry in the background.
`devs`	Allow access to local devices (default).
`nodevs`	Deny access to local devices.
`fg`	Retry in foreground (default).
`hard`	Once the file system is mounted, retry subsequent NFS requests until server responds (default).
`intr`	Permit interrupts for hard mounts (default).
`nocto`	Suppress fresh attributes when opening a file.
`noac`	Suppress attribute and name (lookup) caching.
`nointr`	Ignore interrupts for hard mounts.
`port=`*n*	Set server UCP port number to *n* (default is the port customarily used for NFS servers).

`retrans=`*n*	Set number of NFS retransmissions to *n* (default = 4).
`retry=`*n*	Set number of *mount* failure retries to *n* (default = 1).
`rsize=`*n*	Set read buffer size to *n* bytes (default set by kernel).
`timeo=`*n*	Set NFS timeout to *n* tenths of a second (default = 7).
`wsize=`*n*	Set write buffer size to *n* bytes (default set by kernel).
`soft`	Once the file system is mounted, return error if server does not respond.

Regular defaults are:

`acregmin=3`, `acregmax=60`, `acdirmin=30`, `acdirmax=60`.
`actimeo` has no default; it sets `acregmin`, `acregmax`, `acdirmin`, and `acdirmax` to the value specified.

The `bg` option causes *mount* to run in the background if the server's mount daemon does not respond. *mount* attempts each request `retry=`*n* times before giving up. Once the file system is mounted, each NFS request made in the kernel waits `timeo=`*n* tenths of a second for a response. If no response arrives, the timeout is multiplied by `2` and the request is retransmitted. When `retrans=`*n* retransmissions have been sent with no reply, a `soft` mounted file system returns an error on the request and a `hard` mounted file system retries the request. By default, the `retry` requests for a `hard` mounted file system can be interrupted. If the `nointr` option is specified, `retry` requests for a `hard` mounted file system are not interruptable which means that `retry` requests continue until successful. File systems that are mounted `rw` (read-write) should use the `hard` option. The number of bytes in a read or write request can be set with the `rsize` and `wsize` options. The `devs` option allows access to devices attached to the NFS client via device files located on the mounted NFS file system. The `nodevs` option denies access to devices attached to the NFS client by causing attempts to read or write to NFS device files to return an error.

File Attributes

The attribute cache retains file attributes on the client. Attributes for a file are assigned a time to be flushed. If the file is modified before the flush time, the flush time is extended by the time since the last modification (under the assumption that files that changed recently are likely to change soon). There is a minimum and maximum flush time extension for regular files and for directories. Setting `actimeo=`*n* extends flush time by *n* seconds for both regular files and directories.

NFS Mounts in HP Clustered Environment:

If an NFS mount is made from the server node or any of the client nodes, the NFS file system is accessible from all nodes. If the mount is made from a client node and the client node is rebooted, the mount continues to function normally for all remaining nodes, and will function for the rebooted client without remounting the NFS file system.

DIAGNOSTICS

Attempting to mount a currently mounted non-`nfs` volume under another name results in an error.

umount complains if the special file is not mounted or if it is busy. The file system is busy if it contains an open file or some user's working directory.

Cascaded distributed mounts (mounts whose mount point resides on another client node's locally mounted disk) are not supported.

EXAMPLES

Mount a local disk:

```
mount /dev/dsk/c0d0s4 /usr
```

NFS:

Mount a remote file system:

```
mount serv:/usr/src /usr/src -t nfs
```

Same as above:

```
mount serv:/usr/src /usr/src
```

Same as above but with a soft mount; the file system is mounted read-only:

```
mount serv:/usr/src /usr/src -o soft,ro
```

WARNINGS

A directory or file must be exported by **exportfs** before it is NFS mounted (see *exportfs*(1M)).

Some degree of validation is done on the file system. However, it is generally unwise to mount file systems that are defective, corrupt, or of unknown origin.

Mounting CD-ROM media on a cnode that is not the cluster server is not supported.

AUTHOR

mount was developed by HP, AT&T, the University of California, Berkeley, and Sun Microsystems.

FILES

`/etc/checklist`	file system table
`/etc/mnttab`	mount table

SEE ALSO

fsclean(1M), quotaon(1M), mount(2), checklist(4), mnttab(4), quota(5).

NAME

mountd - NFS mount request server

SYNOPSIS

/usr/etc/rpc.mountd [**-l** *log_file*] [**-t** *n*] [**-e**|**-n**]

DESCRIPTION

mountd is an RPC server that answers file system mount requests. It reads file **/etc/xtab** (described in *exports*(4)) to determine which directories are available to which machines. It also provides information on what file systems are mounted by which clients. This information can be printed using the **showmount** command (see *showmount*(1M)).

rpc.mountd is started from **/etc/netnfsrc** or is invoked by **/etc/inetd** (see *inetd*(1M)).

Options

mountd recognizes the following options:

-l *log_file* Log any errors to the named log file, *log_file*. Errors are not logged if the **-l** option is not specified.

The information logged to the file includes the date and time of the error, the host name, process ID and name of the function generating the error, and the error message. Note that different services can share a single log file since enough information is included to uniquely identify each error.

-e Exit after serving each RPC request. When this option is used, the **inetd** security file **/usr/adm/inetd.sec** can control access to RPC services.

-n Exit only if:

- **portmap** dies (see *portmap*(1M)),
- another **rpc.mountd** registers with **portmap**, or
- **rpc.mountd** becomes unregistered with **portmap**.

This option is more efficient because a new process is not launched for each RPC request. This option is the default.

-t*n*

Specify tracing level *n*, where *n* can have one of the following values:

1 Errors only (default)
2 Errors, mount requests and mount failures

WARNINGS

If a client crashes, executing **showmount** on the server will show that the client still has a file system mounted; i.e., the client's entry is not removed from **/etc/rmtab** until the client reboots and executes **umount -a** (see *showmount*(1M)).

Also, if a client mounts the same remote directory twice, only one entry appears in **/etc/rmtab**. Doing a **umount** of one of these directories removes the single entry and **showmount** no longer indicates that the remote directory is mounted.

If using clustered systems capabilities, only the cluster server's **mountd** can respond successfully to mount requests. In the case of cluster clients, the **rpc.mountd** process is only used to answer **showmount** requests.

AUTHOR

mountd was developed by Sun Microsystems, Inc.

FILES

/etc/rmtab list of all hosts having file systems mounted from this machine

SEE ALSO

inetd(1M), mount(1M), portmap(1M), showmount(1M), exports(4), inetd.conf(4), inetd.sec(4), rmtab(4), services(4).

NAME

mstm - Menu interface to the Support Tools Manager

SYNOPSIS

/usr/diag/bin/mstm [**-m**] [**-l** *log_file*]

DESCRIPTION

mstm provides access to a variety of support actions via a menu-based user interface. **mstm** is a complementary interface to **xstm** (a graphical interface for the X11 environment), and to **cstm** (a command line interface for ASCII terminals). **mstm** can be run on any HP 2392 terminal (or compatible), or on any X terminal.

Before attempting to use the **mstm** interface, be aware of the use of:

select bar
: Selects device classes, devices to test, and help topics. Use arrow keys to move the select bar up and down the menu to the desired item, then select the appropriate function key displayed at the bottom of the screen.

Refresh Screen key
: Useful for refreshing the screen when a modem fault causes incorrect characters to be displayed. Press this key to restore uncorrupted display contents.

Help & Options key
: Access the online help facility to obtain information about a particular system feature or option.

When **mstm** is started, an initial System Mapping screen is displayed. Once system mapping is complete, the Summary Screen is displayed.

Note that when **mstm** is run for the first time, initialization causes a longer wait before user interaction begins than when **mstm** is run again and full initialization is not necessary. This initial execution of **mstm** is the equivalent of using the **-m** option (discussed below).

The Summary Screen shows the following device classes: **SPU**, **Disk**, and **Tape**. From the Summary Screen, the user can choose to run one or more operations on one or more device classes, or on the system as a whole. Currently available operations include verifying the system or verifying a device class. Individual devices can be selected for test by selecting the **List Devices** key displayed at the bottom of the Main Menu screen. When the **List Devices** key is pressed, the Detail Configuration Screen is displayed. Selects **ALL** at the Summary Screen (via the highlight bar) displays a map of the I/O and system configuration, including processors, I/O cards, and peripherals. The map includes the physical path of each device, a brief description of each device, and the status of all support operations that have been initiated on each device. Selecting any of the specific device class from the Summary Screen (such as, **SPU**, **DISKS**, or **TAPES**) displays information about the corresponding subset in the Detail Configuration Screen map information.

The **Help & Options** key displays a brief list of commands available in **mstm**. To obtain additional help information for a particular command, move the highlight bar up and down the list of entries on the Help Screen to the appropriate list item, then selecting the **More Help** function key at the bottom of the screen.

Operations that can be performed on a given specified device depend on the support tool functions available for that type of device, and on the user's security level. Typically, these actions are **verify**, **diagnose**, or **exercise**. **verify** operations consist of tasks that access the device in a manner reflecting a typical user, to determine whether the device is functional. **diagnose** operations perform hardware diagnostic tests on the selected device. **exercise** operations attempt to stress the device.

When an operation such as **verify**, **diagnose**, or **exercise** is selected, a message is displayed indicating that the operation was initiated. When the action has completed, a message is displayed indicating the result of the operation. Additionally, the status displayed on the Summary and Detail Configuration screens is changed to reflect the result. Typical result statuses are **Success**, **Failed**, **Warning**, and **Incompl**. In the case of **Warning**, **Failed**, and **Incompl**, the result is displayed in inverse video in order to make it easier to recognize. If an operation fails, detailed information concerning the exact nature of the failure can be obtained by selecting the **View Logs** key.

To view the log file for the entire session, use the **View Logs** key while the highlight bar is on the **(ALL)** selection on the Summary or Detail Configuration Screens; to view the log file for a specific device,

the highlight bar must be on the device whose log file is to be viewed (assuming any device log file exists — if one does not, an appropriate message is displayed in the Info Area of the screen instructing you to refer to the session log file).

The most current test results are appended to the file. To see the most current test results first, select the `Go end of file` function key, then scroll backwards through the file using the arrow or paging keys. To search on a specific keyword, select the `Search Backward` function key instead. To search in a forward direction instead, use the `Search Forward` key.

Some actions may require user intervention such as mounting a tape and making sure that the tape drive is on line. When such operator intervention is required, `mstm` prompts for a response.

Options

`mstm` recognizes the following options and command-line arguments:

`-m` At start-up, force a search of the physical devices on the system, and of the diagnostic programs supported. When this option is used, `mstm` takes longer to reach the point where it interacts with the user. This option is required when system configuration is changed or when a new diagnostic program is installed through `sysdiag`.

`-l` *log_file*
Specifies the name of the file to which log events are posted, that occur during the time the application is active. The default is `./stm.log`. The contents of this log can be viewed by selecting the `View Logs` key while the highlight bar is on the `(ALL)` selection on the Summary or Detail Configuration Screens.

Since the most current test results are appended to the file, the user should select the `Go end of file` function key then scroll backwards through the file, to see the most current test results first. Use the arrow keys or paging keys to do this, or select the `Search Backward` function key, which will prompt you for a specific keyword to search on. The `Search Forward` key is used when searching forward by keyword.

WARNINGS

Due to the nature of this application and its associated processes, overall system performance may be degraded while `mstm` is running.

AUTHOR

`mstm` was developed by HP.

FILES

Session Log File

The session log (default name: `stm.log`) contains a detailed history of the actions performed by `mstm`. This log begins with a chart indicating the system and I/O configuration for the system and records the results for each action performed during the session. Each line of the chart specifies the location of the device, a description of the device, the current status of the diagnose action, and the current status of the verify action.

Note that the Verifier, Diagnostic, and Exerciser status columns may contain entries of `N/A` or `Not Checked`. `N/A` signifies that the corresponding action is not available for the device. `Not Checked` signifies that the action has not been invoked, which is always the case when `mstm` is first invoked.

Other Files

`/usr/diag/bin/am`	support application manager
`/usr/diag/bin/dtm`	diagnostic tool manager
`/usr/diag/bin/DTMDUI.sh`	diagnostic interface shell
`/usr/diag/bin/diskdaf`	disk device access verifier
`/usr/diag/bin/idiskdaf`	removable-media disk device access verifier
`/usr/diag/bin/tapedaf`	tape device access function
`/usr/diag/bin/CSVER000`	NLS message catalog for the platform
`/usr/diag/bin/MSTM000`	NLS message catalog for MSTM

SEE ALSO

xstm(1M), cstm(1M).

NAME

mvdir - move a directory

SYNOPSIS

`/etc/mvdir` *dir newdir*

DESCRIPTION

`mvdir` moves one directory tree into another existing directory (within the same file system), or renames a directory without moving it.

dir must be an existing directory.

If *newdir* does not exist but the directory that would contain it does, *dir* is moved and/or renamed to *newdir*. Otherwise, *newdir* must be an existing directory not already containing an entry with the same name as the last pathname component of *dir*. In this case, *dir* is moved and becomes a subdirectory of *newdir*. The last pathname component of *dir* is used as the name for the moved directory.

`mvdir` refuses to move *dir* if the path specified by *newdir* would be a descendent directory of the path specified by *dir*. Such cases are not allowed because cyclic sub-trees would be created as in the case, for example, of `mvdir x/y x/y/z/t` which is prohibited.

`mvdir` does not allow directory `.` to be moved.

Only users who have appropriate privileges can use `mvdir`.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

The restriction on names is intended to prevent the creation of cyclic sub-trees that may be inaccessible. `mvdir` checks for such cases strictly by name, thus creating such a sub-tree is still possible. For example, `mvdir x/y x/y/z/t` reports an error, but `mvdir x/y ./x/y/z/t` (effectively the same command) does not, and a cyclic sub-tree results. Be very careful when using the names `.` and `..` while moving directories. It is possible to move `.` by using another name which specifies the current working directory, as in the example, `mvdir ./subdir/.. newdir`.

SEE ALSO

cp(1), mkdir(1), mv(1).

STANDARDS CONFORMANCE

mvdir: SVID2

(Requires Optional ARPA Services Software)

NAME

named - Internet domain name server

SYNOPSIS

named [**-d** *debuglevel*] [**-p** *port_number*] [[**-b**] *bootfile*]

DESCRIPTION

named is the Internet domain name server. See RFC1034 and RFC1035 for more information on the Domain Name System. Without any arguments, *named* reads the default boot file **/etc/named.boot**, reads any initial data, and listens for queries.

Options are:

-d Print debugging information. A number after the **d** determines the level of messages printed.

-p Use a different port number.

-b Use a boot file other than **/etc/named.boot**.

Any additional argument is taken as the name of the boot file. The boot file contains information about where the name server gets its initial data. If multiple boot files are specified, only the last is used. Lines in the boot file cannot be continued on subsequent lines. The following is a small example:

```
;
;       boot file for name server
;
directory       /usr/local/domain

; type      domain                  host/file       backup file

cache       .                       db.cache
primary     berkeley.edu            db.berkeley
primary     32.128.in-addr.arpa     db.128.32
secondary   cc.berkeley.edu         128.32.137.8 db.cc
secondary   6.32.128.in-addr.arpa   128.32.137.8 db.128.32.6
primary     0.0.127.in-addr.arpa    db.127.0.0
forwarders  10.0.0.78 10.2.0.78
; slave
sortlist    10.0.0.0 26.0.0.0
```

Comments in the boot file start with a ; and end at the end of the line. Comments can start anywhere on the line.

The **directory** line causes the server to change its working directory to the directory specified. This can be important for the correct processing of $INCLUDE files (described later) in primary servers' master files. There can be more than one **directory** line in the boot file if master files are in separate directories.

Files referenced in the boot file contain data in the master file format described in RFC1035.

A server can access information from servers in other domains given a list of root name servers and their addresses. The **cache** line specifies that data in **db.cache** is to be placed in the backup cache. Its use is to prime the server with the locations of root domain servers. This information is used to find the current root servers and their addresses. The current root server information is placed in the operating cache. Data for the root nameservers in the backup cache are never discarded. There can be more than one **cache** file specified.

The first **primary** line states that the master file **db.berkeley** contains authoritative data for the **berkeley.edu** zone. A server authoritative for a zone has the most accurate information for the zone. All domain names are relative to the origin, in this case, **berkeley.edu** (see below for a more detailed description). The second **primary** line states that the file **db.128.32** contains authoritative data for the domain **32.128.in-addr.arpa**. This domain is used to translate addresses in network **128.32** to hostnames. The third **primary** line states that the file **db.127.0.0** contains authoritative data for the domain **0.0.127.in-addr.arpa**. The domain is used to translate **127.0.0.1** to the name used by the loopback interface. Each master file should begin with an SOA record for the zone (see below).

The first **secondary** line specifies that all authoritative data in the **cc.berkeley.edu** zone is to be transferred from the name server at Internet address 128.32.137.8 and will be saved in the backup file

db.cc. Up to 10 addresses can be listed on this line. If a transfer fails, it will try the next address in the list. The secondary copy is also authoritative for the specified domain. The first non-Internet address on this line will be taken as a filename in which to backup the transferred zone. The name server will load the zone from this backup file (if it exists) when it boots, providing a complete copy, even if the master servers are unreachable. Whenever a new copy of the domain is received by automatic zone transfer from one of the master servers, this file is updated.

The **forwarders** line specifies the addresses of sitewide servers that will accept recursive queries from other servers. If the boot file specifies one or more forwarders, then the server will send all queries for data not in the cache or in its authoritative data to the forwarders first. Each forwarder will be asked in turn until an answer is returned or the list is exhausted. If no answer is forthcoming from a forwarder, the server will continue as it would have without the forwarders line unless it is in **slave** mode. The forwarding facility is useful to cause a large sitewide cache to be generated on a master, and to reduce traffic over links to outside servers.

The **slave** line (shown commented out) is used to put the server in slave mode. In this mode, the server will only make queries to forwarders. This option should only be used in conjunction with using forwarders.

The **sortlist** line can be used to indicate networks that are preferred over other, unlisted networks. Address sorting only happens when the query is from a host on the same network as the server. The best address is placed first in the response. The address preference is local network addresses, then addresses on the sort list, then other addresses.

The master file consists of control information and a list of resource records for objects in the zone of the forms:

$INCLUDE <*filename*> <*opt_domain*>
$ORIGIN <*domain*>
<*domain*> <*opt_ttl*> <*opt_class*> <*type*> <*resource_record_data*>

where *domain* is . for root domain, @ for the current origin (where current origin is the domain from the boot file or the origin from an $ORIGIN line), or a standard domain name. If *domain* is a standard domain name that does not end with ., the current origin is appended to the domain. Domain names ending with . are unmodified. The *opt_domain* field is used to define an origin for the data in an included file. It is equivalent to placing a $ORIGIN statement before the first line of the included file. The field is optional. Neither the *opt_domain* field nor $ORIGIN statements in the included file modify the current origin for this file. The *opt_ttl* field is an optional integer number for the time-to-live field. It defaults to zero, meaning the minimum value specified in the SOA record for the zone. The *opt_class* field is the object address type; currently only two types are supported: **IN** and **HS**. **IN** is for objects connected to the DARPA Internet and **HS** is for objects in the Hesiod class. The *type* field contains one of the following tokens; the data expected in the *resource_record_data* field is in parentheses:

A	A host address (dotted quad)
CNAME	Canonical name for an alias (domain)
GID	Group ID (integer)
HINFO	host information (*cpu_type* OS_type)
MB	a mailbox domain name (domain)
MG	a mail group member (domain)
MINFO	mailbox or mail list information (*request_domain error_domain*)
MR	a mail rename domain name (domain)
MX	a mail exchanger (domain)
NS	an authoritative name server (domain)
PTR	a domain name pointer (domain)
SOA	marks the start of a zone of authority (domain of originating host, domain address of maintainer, a serial number and the following parameters in seconds: refresh, retry, expire and minimum TTL)

TXT	text data (string)
UID	user ID (integer)
UINFO	user information (string)
WKS	a well known service description (IP address followed by a list of services)

Not all of the data types are used during normal operation. The following data types are for experimental use and are subject to change: GID, MB, MG, MR, UID, UINFO.

Resource records normally end at the end of a line, but can be continued across lines between opening and closing parentheses. (The continuation is currently only implemented for SOA and WKS records.) Comments are introduced by semicolons and continue to the end of the line.

Each master zone file should begin with an SOA record for the zone. An example SOA record is as follows:

```
@       IN      SOA     ucbvax.berkeley.edu. root.ucbvax.berkeley.edu. (
                                89              ; Serial
                                10800   ; Refresh every 3 hours
                                3600            ; Retry every hour
                                604800 ; Expire after a week
                                86400 ); Minimum ttl of 1 day
```

The SOA lists a serial number, which should be increased each time the master file is changed. Secondary servers check the serial number at intervals specified by the refresh time in seconds; if the serial number increases, a zone transfer is done to load the new data. If a master server cannot be contacted when a refresh is due, the retry time specifies the interval at which refreshes should be attempted until successful. If a master server cannot be contacted within the interval given by the expire time, all data from the zone is discarded by secondary servers. The minimum value is the time-to-live used by records in the file with no explicit time-to-live value.

The following signals have the specified effect when sent to the server process using the *kill*(1) command:

SIGHUP	Causes server to read the boot file and reload database.
SIGINT	Dumps current data base and cache to **/usr/tmp/named_dump.db**.
SIGIOT	Dumps statistics data into **/usr/tmp/named.stats**. Statistics data is appended to the file.
SIGUSR1	Turns on debugging; each SIGUSR1 increments debug level.
SIGUSR2	Turns off debugging completely.

sig_named(1M) can also be used for sending signals to the server process.

DIAGNOSTICS

Any errors encountered by *named*(1m) in the boot file, master files, or in normal operation are logged with syslog and in the debug file, **/usr/tmp/named.run**, if debugging is on.

AUTHOR

named was developed by the University of California, Berkeley.

FILES

/etc/named.boot	name server configuration boot file
/etc/named.pid	process ID
/usr/tmp/named.run	debug output
/usr/tmp/named_dump.db	dump of the name server database
/usr/tmp/named.stats	nameserver statistics data

SEE ALSO

kill(1), hosts_to_named(1M), sig_named(1M), signal(2), gethostent(3N), resolver(3N), resolver(4), hostname(5), RFC1032, RFC1033, RFC1034, RFC1035.

NAME

ncheck - generate path names from inode numbers

SYNOPSIS

/etc/ncheck [**-i** *inode-numbers*] [**-a**] [**-s**] [*file-system*]

DESCRIPTION

ncheck, when invoked without arguments, generates a list of path names corresponding to the inode numbers of all files in volumes specified by **/etc/checklist**. Path names generated by **ncheck** are relative to the given file system. Names of directory files are followed by **/**.

Options

-i *inode-numbers*
: Report only on files whose inode numbers are specified on the command line in the *inode-numbers* list.

-a
: Allow printing of the names **.** and **..**, which are ordinarily suppressed.

-s
: Report only on special files and regular files with set-user-ID mode. (Context dependent files, which are set-user-ID directories, are not included in the report.) The **-s** option is intended to discover concealed violations of security policy.

A *file-system* (block special file) can be specified.

The report is in no useful order, and probably should be sorted.

Context-Dependent Files

ncheck appends a plus sign (+) after files that are hidden directories (context-dependent files). See *cdf*(4).

Access Control Lists

Continuation inodes (that is, inodes containing additional access control list information) are quietly skipped since they do not correspond to any path name.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

When the file system structure is improper, **??** denotes the "parent" of a parentless file and a path-name beginning with **...** denotes a loop.

WARNINGS

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

AUTHOR

ncheck was developed by AT&T and HP.

SEE ALSO

sort(1), fsck(1M), cdf(4), checklist(4), acl(5).

STANDARDS CONFORMANCE

ncheck: SVID2

NAME

netdistd - network file distribution (update) server daemon

SYNOPSIS

/etc/netdistd [**-C** *connections*] [**-f** *file*] [**-l**] [**-L** *logfile*] [**-P** *port*] [**-v**] [**-c** *directory*]

DESCRIPTION

netdistd is the server daemon for the "netdist" file distribution service. The netdist service supports the distribution of packages, basically collections of files, from a server system (host) to a client system (host). Currently, **update** is available as a netdist client (see *update*(1M)).

Options

netdistd recognizes the following options:

-C *connections*
: Set maximum allowable number of simultaneous connections (default is 20).

-f *file*
: Use *file* as the central package definition file instead of the default file (see FILES below). Use this option to "connect" **netdistd** to a non-default directory tree.

-l
: Append event information to the default log file (see FILES below). The **netdistd** command does no logging unless this option or **-L** is specified. **netdistd** seeks to the end of the log file before each write to it, making it safe to truncate the logfile while the daemon is running.

-L *logfile*
: Activate logging but use *logfile* instead of the default log file. If *logfile* is **-**, **netdistd** runs in the foreground (does not detach from the display), and logging goes to standard output.

-P *port*
: Specify the port number to use instead of the default port number associated with the netdist service in the network services file (see FILES below).

-v
: Trace package execution (verbose logging). This option is ignored unless **-l** or **-L** is also specified. Repeating **-v** up to 6 times increases the relative amount of verbosity.

-c *directory*
: Specify a directory to hold temporary subdirectories for caching information files for efficient transfer to **update**. By default the temporary subdirectories reside in the directory that contains the central package definition file. Specify *directory* as **none** to turn off the **update** information-file caching feature.

Log Files

Each line in a log file begins with one of two kinds of identifiers:

netdistd *.pid*
counter .pid

Lines of the first form are emitted by the originally scheduled **netdistd** daemon and contain its process ID number. **netdistd** starts a new child process to handle each connection (service request). Lines of the second form are emitted by child processes and contain their process ID numbers. *counter* is a number incremented for each child process started by a server.

Control Signals

Once **netdistd** is running, you can send it one of the following signals using **kill** to change its state (see *kill*(1)):

SIGINT **SIGPIPE** **SIGTERM**
: If caching is enabled and any child processes are running (that is, connections spawned by this server are active), response to these signals (that is, termination of the parent server process) is delayed until all children terminate, and no new connections to the server are allowed until this occurs.

SIGHUP
: Re-read the package definition files. If logging is enabled, close and reopen the log file. If caching is enabled, remove and recreate the cached information files, but if any child processes of this server are running, all response to the signal is delayed until all of them terminate, and no new connections are allowed to this server until this occurs.

SIGQUIT
: Toggle logging off and on.

`SIGUSR1` Increase the trace level by one.

`SIGUSR2` Decrease the trace level by one.

Package Definition Files

Files available through the netdist service are bundled into "filesets" (see *update*(1M)). Each fileset has an associated "package definition file" stored in a fileset-dependent location (see FILES below). The package definition file is a `netdistd` control file that, among other things, specifies which files belong to the fileset. In addition, a central package definition file identifies the filesets that are available for distribution, and contains pointers to individual fileset package definition files.

When performing a network file distribution, `netdistd` reads the central package definition file (see FILES below) to determine what filesets are available from the server system and to obtain the pointer to the package definition file for each fileset to be distributed.

Temporary Subdirectories and Cached Files

The `netdistd` program creates temporary subdirectories containing cached `update` information files. By default the subdirectories are located under the directory containing the central package definition file.

This feature allows `update` and `updist` clients to get fileset information very quickly (see *updist*(1M)).

The temporary subdirectories and cached files are automatically removed when the `netdistd` parent process stops or is terminated. They are automatically removed and recreated when the `netdistd` parent process receives a `SIGHUP` signal, as part of re-reading the package definition files.

The subdirectories are named `tmp`*port*`.`*arch*, where *port* is the port number the server uses and *arch* is the architecture type of the filesets to which the cached information files correspond. For example, a `netdistd` program using port 2106 and serving Series 300/400, Series 700, and Series 800 filesets creates three subdirectories called `tmp2106.300`, `tmp2106.700`, and `tmp2106.800`.

Cached files consume about 700 bytes of disk space for each fileset served plus about 60 bytes of disk space for each file served. Typically this adds up to about 4Kbytes of temporary disk space for each fileset served. Viewed another way, the space used is about 0.3% of the total disk space for the served filesets.

If caching is disabled (see the `-c` option above), less disk space is used, but client processes take longer to get fileset information.

Setting Up a Network Distribution Server

When setting up a system to support the netdist service, follow these steps:

1. Use `updist` to load the desired filesets onto the server system. The recommended destination directory under which to load the filesets is the default, `/netdist`.

   Directly below the destination directory, `updist` creates a central package definition file (see FILES below) containing a **source** statement for each distributable fileset. Commenting or uncommenting the **source** statements affects which filesets the server program can distribute (lines beginning with # are treated as comments in the usual manner).

   Note: `rmfn` cannot remove netdist packages. To remove a package (fileset), edit the central package definition file to comment out or delete references to the deleted package, and send `netdistd` a `SIGHUP` signal. Then, if desired, run `rm -r` on the package's directory under the `updist` destination directory to remove the package's files. **Warning**: Do not modify or delete the package's files until all server child processes that might be using them complete.

2. Ensure that there is an entry for the netdist service in the networking services definition file (see FILES below). For example:

   ```
   netdist  2106/tcp  # network file distribution
   ```

3. Start the `netdistd` daemon by invoking it with any options desired. The `netdistd` command automatically starts a background process and returns control to the caller. `netdistd` can also be called from system start up scripts, but do not call it from `init` directly, because it appears to terminate immediately, possibly causing `init` to respawn it (see *init*(1M)).

4. The server system is now ready for use by `update`. To verify, run `update` interactively and specify the server system's hostname as the update source. Verify that the available filesets correspond to those

loaded on the server system. Another verification method is to run:

```
update -c -s hostname
```

on the server system and look for the desired filesets in the output.

Note: If the source is a netdist server system, `update -c` lists only those available filesets that match the client system's architecture type. Use the `update -S` option to list the other type. For example, on a Series 800 system:

```
update -c -s hostname -S300
```

5. Set permissions for remote systems to access the network server (see Security section below).

Modifying a Network Distribution Server

The safest method is:

1. `tail -f` *logfile* `&` This starts a continuous tail of the log file to the screen so you can see when `netdistd` terminates.
2. `kill` *parent-server* If clients are connected, the `netdistd` parent process delays terminating until any existing child processes terminate, and meanwhile new connections are refused.
3. Wait for the `netdistd` server to stop, watching the previously started `tail` of the *logfile*. Clients doing a full operating system update, and clients just sitting there with a selection screen (possibly forgotten by their creators), could make you wait a long time.
4. Add, modify, or delete filesets in the netdist tree.
5. Restart the `netdistd` server.
6. Kill the `tail` process you started in the background.

The riskier method of modifying a netdist server tree is:

1. Add, modify, or delete filesets in the netdist tree. (Only adding filesets is completely safe using this method.)
2. `kill -1` *parent-server* If clients are connected, the `netdistd` parent process delays rereading the package description file and rebuilding cached files until all existing child processes terminate. Meanwhile, new connections are refused.

Security

Whenever the `netdistd` daemon receives a connection request, it performs a security check using the same mechanism as `inetd` (see *inetd.sec*(4)).

RETURN VALUE

The `netdistd` command returns 0 if it encounters no errors; 1 otherwise.

DIAGNOSTICS

When invoked, `netdistd` parses and compiles the instructions contained in the package definition files. It then does an access check on the source files to be distributed. If an error occurs, `netdistd` prints an error message to the log file and halts.

EXAMPLES

Run `netdistd` with default connections limit, default central package definition file, default port number, and no logging:

```
netdistd
```

Run `netdistd` with a maximum of two simultaneous connections, an alternate central package definition file, very verbose logging to an alternate log file, and a special port number:

```
netdistd -C2 -f /netd2/MAIN.pkg -vvL /tmp/netdistd.log -P 2111
```

DEPENDENCIES

Series 700

For HP-UX release 8.01, there is little externally visible difference between Series 700 and Series 800 update media sub-tree names and structure. Consequently, if `updist` is used to create netdist servers for both Series 700 and Series 800 filesets, the filesets must be placed in separate destinations (netdist trees)

so they do not overwrite each other. This can be done by, for example, sending the Series 700 filesets to a destination of `/netdist.700`. Files from Series 700 update media are then placed in a directory subtree named `800` under directory `/netdist.700`. Run separate `netdistd` programs on different ports for each tree. For example, to start a server for the `/netdist.700` tree:

```
netdistd -f /netdist.700/MAIN.pkg -P 2107
```

For 8.05 and later releases of HP-UX, a separate destination (netdist tree) is not necessary. `updist` deposits filesets in a sub-tree under the netdist tree for each system type.

AUTHOR

`netdistd` was developed by HP.

FILES

`/netdist/MAIN.pkg`	default central package definition file
`/netdist/300/`*fileset-name*`/netdist.pkg`	Series 300/400 package definition files
`/netdist/700/`*fileset-name*`/netdist.pkg`	Series 700 package definition files
`/netdist/800/`*fileset-name*`/netdist.pkg`	Series 800 package definition files
`/netdist/tmp2106.300`	
`/netdist/tmp2106.700`	
`/netdist/tmp2106.800`	default temporary subdirectories for cached `update` information files
`/usr/adm/netdist.log`	default log file
`/etc/services`	networking services definition file; contains default port number
`/usr/adm/inetd.sec`	`inetd` security file also read by `netdistd`

SEE ALSO

kill(1), inetd(1M), init(1M), rmfn(1M), update(1M), services(4), inetd.sec(4).

NAME

netfmt - format tracing and logging binary files.

SYNOPSIS

/etc/netfmt [**-I** *subsys_file*] [**-c** *config_file* [**-p**]] [**-F**] [**-t** *records*] [**-v**] [**-l**] [**-n**] [**-N** | [**-l** [**LT**]]] [[**-f**] *file_name*]

DESCRIPTION

netfmt is used to format binary trace and log data gathered from the tracing and logging facility. The binary trace and log information can be read from a file or from standard input (if standard input is a tty device, an informative message is given and **netfmt** quits). Formatted data is written to standard output. Formatting options are specified in an optional filter configuration file. Message inclusion and format can be controlled by the filter configuration file. If no configuration commands are specified, all messages are fully formatted. A description of the filter configuration file follows the option descriptions.

Options

netfmt recognizes the following command-line options and arguments:

-I *subsys_file* Specifies the file containing a description of all subsystems; the option processing and formatting functions to call and the library that contains them. This option may be used to specify an alternate subsystem file configuration file during development of new subsystems and subsystem formatters. If omitted, the default file **/etc/conf/nettlgen.conf** is read to provide this information.

-c *config_file* Specifies the file containing formatter filter configuration commands. Syntax for the commands is given below. When **-c** is omitted the file **$HOME/.nettrc** (for trace files) or **$HOME/.netlogrc** (for log files) is read for filter configuration commands if it exists.

-p Parse input: this switch allows the user to perform a syntax check on the *config_file* specified by the **-c** parameter. All other parameters are ignored. If the syntax is correct, **netfmt** terminates with no output or warnings.

-F Follow the input file. Instead of closing the input file when end of file is encountered, **netfmt** keeps it open and continues to read from it as new data arrives. This is especially useful for watching events occur in real time while troubleshooting a problem. Another use would be for recording events to a console or hard-copy device for auditing. (Note that console logging is controlled by the configuration files **/etc/conf/nettlgen.conf** and **/usr/adm/conslog.opts**; see *nettlgen.conf*(4).)

-t *records* Specifies the number of records from the tail end of the file to format. This allows the user to bypass extraneous information at the beginning of the file, and get to the most recent information quickly. The maximum number of *records* that can be specified is 1000. If omitted, all records are formatted.

-f *file_name* Specifies the file containing the binary log or trace data. If omitted, data is read from standard input.

-v Enables output of **netfmt** internal debugging information to standard error.

The following options are for LAN and X.25 trace formatting to allow backward compatibility with the obsolete *nettrfmt* command. These options are ignored for all other subsystems.

-l *(ell)* Turn off inverse video highlighting of certain traced fields. Use this flag when sending formatted trace data to a line printer. This option is valid for both LAN and X.25 traces. **Default:** Certain fields in the trace file are highlighted in inverse video when viewing the formatted trace format at a terminal that supports highlighting.

-1 *(one)* Attempts to tersely format each traced packet on a single line. If **-L** and/or **-T** options are used, the output lines will be more than 80 characters long. This option is ignored when formatting X.25 trace data.

-N Enables "nice" formatting where Ethernet/IEEE802.3, SLIP, IP, ICMP, TCP, UDP, PXP, ARP, and Probe packets are displayed symbolically. All remaining user data is

formatted in hexadecimal and ASCII. This option is ignored when formatting X.25 trace data.

`-n` Shows network addresses and ports as numbers (normally, `netfmt` interprets addresses and attempts to display them symbolically). This option is ignored when formatting X.25 trace data.

`-T` Places a time stamp on tersely formatted packets. Used with the `-1` (*minus one*) option. This option is ignored when formatting X.25 trace data.

`-L` Prefixes local link address information to terse tracing output. Used with the `-1` (*minus one*) option. This option is ignored when formatting X.25 trace data.

Filter Configuration File

Note: Filter configuration file syntax converges the syntax used with the obsolete `nettrfmt` network trace formatter and `netlogfmt` network log formatter commands with new `netfmt` syntax for controlling formatter options. The first section below describes the general use and syntax of the filter configuration file. Specific options for LAN and X.25 Naming and Filtering are listed in the *Subsystem Filtering* section.

The configuration file allows specification of two types of information:

- IEEE802.3/Ethernet-address to node-name mapping causes `netfmt` to substitute the specified node name for its address in the output; this is only available on the LAN tracing output as described in later sections.
- Specify filters in order to precisely tailor which packets are to be discarded and which are to be formatted. **Global filters** control all subsystems; **subsystem filters** pertain only to specific subsystems.

A filter is compared against values in input packets. If a packet matches a filter, the packet is formatted; otherwise, the packet is discarded. A filter can also specify *NOT* by using `!` before the filter value in the configuration file. If a packet matches a *NOT* filter, the packet is discarded. A filter can also be a "wildcard" (matching any value) by specifying an asterisk `*` before the filter value in the configuration file. "Wild card" filters pass all packets of the specified protocol layer. Specifying `!*` as the filter means *NOT ALL*. This indicates not all, and is usually followed by specifications to turn on filters for specific messages. This must be used in conjunction with other filters of the same type to actually include something in the formatted output.

Configuration File Syntax

- The host formatter ignores white space, such as spaces or tabs. However, newlines (end of line characters) are important, as they terminate comments and specifications.
- The formatter is not case sensitive. For example `error` and `ERROR` are treated as equivalent.
- To place comments in the file, begin each comment line with a `#` character. The formatter ignores all remaining characters on that line.
- An exclamation point (`!`) in front of an argument indicates *NOT*. This operator is not supported for timestamp, log instance, and ID filtering.
- The asterisk (`*`), when used as an argument, indicates *ALL*. Since the default for all formatting options is *ALL*, it is unnecessary to use the asterisk alone. It can be used along with the exclamation point, (`!*`) to indicate *NOT ALL*. This operator is not available for timestamp, log instance, and ID filtering.

Global Filtering:

Six types of global filtering are provided:

`class`	log class: `Disaster`, `Error`, `Warning`, `Informative`
`kind`	trace kind
`id`	connection, process, path, and user
`log instance`	specific thread of events
`subsystem`	see *nettl*(1M) or use `nettl -status all` command for a list of subsystems
`time`	specify ranges of time(s)

config_file entry syntax: Global filtering specifications are indicated by the words `formatter filter` followed by type and value information in the form:

formatter filter *type* [**!**] *value* | *****

The following combinations are recognized:

formatter filter class *value*

value indicates the log class. Initially all log classes are permitted. However, by turning off all log classes with the **!*** operator then giving a single class, a specific log class can be formatted.

INFORMATIVE	Describes routine operations and current system values.
WARNING	Indicates abnormal events possibly caused by subsystems problems.
ERROR	Signals an event or condition which was *not* affecting the overall subsystem or network operation, but may have caused an application program to fail.
DISASTER	Signals an event or condition which *did* affect the overall subsystem or network operation, caused several programs to fail or the entire node to shut down.

formatter filter Connection_ID *value*
formatter filter Path_ID *value*
formatter filter Process_ID *value*
formatter filter User_ID *value*

value specifies the ID number of the messages to format. *Only one id_type is allowed per configuration file*. The **!** operator is *not* allowed in *value*.

formatter filter kind *value*

value can either be an established trace kind or a mask. A mask is a hexadecimal representation of a (set of) trace kind(s). Trace kinds and their corresponding masks are:

Name	Mask	Name	Mask
error	0x02000000	**pduin**	**0x20000000**
hdrin	0x80000000	**pduout**	**0x10000000**
hdrout	0x40000000	**proc**	**0x08000000**
logging	0x01000000	**state**	**0x04000000**
loopback	0x00800000		

formatter filter log_instance *value*

value specifies the log instance number of the messages to filter. Selecting a log instance allows the user to see the messages from a single thread of network events. Only one log instance is allowed per configuration file. The log instance cannot be negated with the **!** operator.

formatter filter subsystem *value*

value specifies the subsystem name. Available subsystem names are listed in the *nettl*(1M) manual entry; they can also be listed by using the command:

```
nettl -status all
```

Only one subsystem name is allowed per line; multiple lines OR the request. To eliminate a given subsystem name, use the **!** operator, which formats all subsystems except those excluded by the list of negated subsystems. To include all subsystems, use the ***** operator (the default). To eliminate all subsystems, use the **!*** operator. Initially all subsystems are enabled for formatting. To format only specific subsystems, turn off all subsystems by using the **!*** operator, then selectively enable the desired subsystems.

formatter filter time_from *value*
formatter filter time_through *value*

time_from indicates the inclusive starting time; **time_through** indicates the inclusive ending time. *value* consists of two fields: *time_of_day* and *day_of_year*, (usually separated by one or more blanks for readability).

time_of_day specifies the time on the 24-hour clock in hours, minutes, seconds and decimal parts of a second (resolution is to the nearest microsecond). Hours, minutes and seconds are required; fractional seconds are optional. *time_of_day* format is *hh* : *mm* : *ss* . *dddddd*.

day_of_year specifies the day of the year in the form month/day/year in the format: *mm/dd/yy*. Specify month and day numerically numerically, using one or two digits. For example, January can be specified as `1` or `01`; the third day of the month as `3` or `03`. Specify the year by its last two digits. For example, specify 1985 as `85`.

The `time_from` specification includes *only* those records after the resolution of time given. For example, if the *time_of_day* for `time_from` is specified as 10:08:00, all times before that, from 10:07:59.999999 and earlier, are excluded from the formatted output. Records with times of 10:08:00.000000 and later are included in the formatted output. Similarly, the `time_through` specification includes *only* up to the resolution of time given. For example, if the *time_of_day* for `time_through` is specified as 10:08:00, all records with times after that, from 10:08:00.000001 onward, are excluded from the formatted output.

The *time_of_day* and *day_of_year* fields are both required.

Subsystem Filtering

Note: Global filtering described above takes precedence over individual subsystem tracing and logging filtering described below.

Subsystem filters are provided to allow filtering of data for individual subsystems or groups of subsystems. Currently, two subsystem filters are provided: LAN/X.25 and OTS. The collection of LAN and X.25 subsystems use the subsystem filters identified by the `FILTER` keyword and the collection of OTS subsystems use the subsystem filters with the `OTS` keyword.

It is possible for each subsystem configured on the system to have an individual subsystem filter. Such a subsystem filter would have the subsystem name as the keyword and would be configured by the `nettlconf` command (see *nettlconf*(1M)). A group of subsystems can share a subsystem filter if the same options processing function and the same subformatter library are configured in the `nettlgen.conf` file for every subsystem in that group (see *nettlgen.conf*(4)). This enables groups of subsystems, such as the LAN/X.25 and OTS subsystems, to use the same subsystem filer.

Subsystem filters are valid only when the corresponding subsystems have been installed and configured on the system. The syntax for individual subsystem formatters is given below. **LAN and X.25 Naming and Filtering**

There are numerous filter types, each associated with a particular protocol layer:

Filter Layer	Filter Type	Description
Layer 1	`dest`	hardware destination address
	`source`	hardware source address
	`interface`	software network interface
Layer 2	`ssap`	IEEE802.2 source sap
	`dsap`	IEEE802.2 destination sap
	`type`	Ethernet type
Layer 3	`ip_saddr`	IP source address
	`ip_daddr`	IP destination address
Layer 4	`tcp_sport`	TCP source port
	`tcp_dport`	TCP destination port
	`pxp_sport`	PXP source port
	`pxp_dport`	PXP destination port
	`udp_sport`	UDP source port
	`udp_dport`	UDP destination port
	`connection`	a level 4 (TCP, UDP, PXP) connection

Filtering occurs at each of the four layers. If a packet matches any filter within a layer, it is passed up to the next layer. The packet must pass every layer to pass through the entire filter. Filtering starts with Layer 1 and ends with Layer 4. If no filter is specified for a particular layer, that layer is "open" and all packets pass through. For a packet to make it through a filter layer which has a filter specified, it must match the filter.

config_file entry syntax: LAN and X.25 filtering specifications take two forms:

`name` *nodename value*

nodename is a character string. *value* is a hardware address consisting of 6 bytes specified in hexadecimal (without leading "0x"), optionally separated by -.

filter *type* [**!**] *value* | *****

filter is the keyword identifying the filter as a LAN/X.25 subsystem filter. The following combinations are recognized:

filter connection *value*
value takes the form:

local_addr **:** *port remote_addr* **:** *port*

where *local_addr* and *remote_addr* can be a hostname or a 4-byte Internet address specified in decimal dot notation (see *inet*(3N) for more information on Internet addresses and decimal dot notations). *port* can be a service name or an *integer*. *integer* represents a port and can be designated by a hexadecimal integer (**0x***digits*), an octal integer (**0***digits*), or base-10 integers (0 through 65 535).

filter dest *value*
filter source *value*
value is a hardware address consisting of 6 bytes specified in hexadecimal (without leading **0x**), optionally separated by **-**.

filter dsap *value*
filter ssap *value*
value is a hexadecimal integer of the form: **0x***digit*; an octal integer of the form: **0***digits*; or a base-ten integer, 0 through 255.

filter interface *value*
value identifies a network interface and takes the form: **lan***n* for LAN interface, or **lo***n* for loopback interface, where *n* is the logical unit number, as in **lan0**.

filter ip_daddr *value*
filter ip_saddr *value*
value is a hostname or a 4-byte Internet address specified in decimal dot notation (see *inet*(3N) for more information on Internet addresses and decimal dot notations).

filter pxp_dport *value*
filter pxp_sport *value*
filter tpc_dport *value*
filter tpc_sport *value*
filter udp_dport *value*
filter udp_sport *value*
value is a port number as a 2-byte integer value or a service name. The integer value can be designated by a hexadecimal integer (0x*digits*), an octal integer (0*digits*), or a base-10 integer (0 through 65 535).

filter subsystem *value*
value takes the form:

subsys_name event event_list

where *subsys_name* can be one of the following:

bufs	**lan**	**nftdaemon**	**pxp**	**tcp**
ip	**nfs**	**nsdiag**	**rlbdaemon**	**udp**
ipc	**nft**	**probe**	**sockregd**	**x25**

event_list takes the form:

event_spec [**,** *event_spec*...]

where *event_spec* takes one of the three forms:

[**!**]*integer* [**!**]*range* [**!**]*****

integer is an integer in hexadecimal (leading **0x**), octal (leading **0**), or decimal, which specifies an event for the subsystem indicated.

range takes the form *integer*-*integer*, and indicates an inclusive set of events.

`filter type` *value*
value is a hexadecimal integer of the form: `0x`*digits*; an octal integer of the form: `0`*digits*; or a base-ten integer, 0 through 65 535.

OTS Naming and Filtering
The OTS subsystem filter allows filtering of the message ID numbers that are typically found in the data portion of an OTS subsystem's log or trace record. The OTS subsystem filter is effective for any subsystem that is a member of the OTS subsystem group.

OTS trace filtering configuration commands have the following form in *config_file*:

`OTS` [*subsystem*] `msgid` [`!`] *message_ID* | `*`

Keywords and arguments are interpreted as follows:

`OTS` Identifies the filter as an OTS subsystem filter.

subsystem One of the following group of OTS subsystems:

`OTS`	`ACSE_PRES`	`NETWORK`
`TRANSPORT`	`SESSION`	

Note: The absence of *subsystem* implies that the filter applies to all OTS subsystems.

message_ID is the value of the message ID to filter. A message ID is used by OTS subsystems to identify similar types of information. It can be recognized as a 4 digit number contained in brackets ([ ]) at the beginning of an OTS subsystem's trace or log record. Initially all *Message_IDs* are enabled for formatting. To format records with specific *Message_IDs*, turn off all message IDs using the `!*` operator, then selectively enable the desired message IDs. Only one *Message_ID* is allowed on each line.

EXTERNAL INFLUENCES
International Code Set Support
Single- and multi-byte character code sets are supported in data. Single-byte character codesets are supported in filenames.

DEPENDENCIES
X.25
The following options are not recognized when formatting X.25 traces: `-1` (*one*), `-N`, `-n`, `-T`, and `-L`.

WARNINGS
The syntax that was used for the obsolete LAN and X.25 trace and log options has been mixed with the syntax for the `netfmt` command such that any old options files can be used without any changes. The combination of syntax introduces some redundancy and possible confusion. The global filtering options have the string `formatter filter` as the first two fields, while the LAN and X.25 filtering options merely have the string `filter` as the first field. It is expected that the older LAN and X.25 filtering options may change to become more congruent with the global filtering syntax in future releases.

The `nettl` and `netfmt` commands read the `/etc/conf/nettlgen.conf` file each time they are executed. If the file becomes corrupted, these commands become no longer operational (see *nettl*(1M) and *netfmt*(1M)).

DIAGNOSTICS
Messages describe illegal use of `netfmt` command and unexpected EOF encountered.

EXAMPLES
The first group of examples show how to use command line options.

1. Read file `/usr/adm/trace.TRC1` for binary data and use `conf.file` as the filter configuration file:

```
netfmt -c conf.file -f /usr/adm/trace.TRC1
```

2. Format the last 50 records in file `/usr/adm/nettl.LOG00` (the default log file):

```
netfmt -t 50 -f /usr/adm/nettl.LOG00
```

3.
Use the follow option to send *all* log messages to the console (normally, **DISASTER**-class log messages are sent to the console in terse form):

```
netfmt -f  /usr/adm/nettl.LOG00  -F  > /dev/console
```

4.
Display all log messages in the **hpterm** window:

```
hpterm -e /etc/netfmt -F -f /usr/adm/nettl.LOG00
```

The remaining examples show how to format entries in the configuration file used with the **-c** option.

1. Tell **netfmt** to format only **INFORMATIVE**-class log messages coming from the **NS_LS_IP** subsystem between 10:31:53 and 10:41:00 on 23 November 1993.

```
formatter       filter       time_from       10:31:53     11/23/93
formatter       filter       time_through    10:41:00     11/23/93
formatter       filter       class           !*
formatter       filter       class           INFORMATIVE
formatter       filter       subsystem       !*
formatter       filter       subsystem       NS_LS_IP
```

2.
Map hardware address to name:

```
name            node1          08-00-09-00-0e-ca
name            node3          02-60-8c-01-33-58
```

3.
Format only packets from either of the above hardware addresses:

```
filter          source         08-00-09-00-0e-ca
filter          source         02-60-8c-01-33-58
```

4.
Format all packets transmitted from the local node to **node1** which reference local TCP service ports **login** or **shell**, or remote UDP port **777**. The local hostname is **local**:

```
filter          ip_saddr       local
filter          ip_daddr       node1
filter          tcp_sport      login
filter          tcp_sport      shell
filter          udp_dport      777
```

5.
Format a TCP connection from local node **node2** to **node1** which uses **node2** service port **ftp** and remote port **1198**.

```
filter          connection     node2:ftp      node1:1198
```

6.
Format all packets except those that use interface **lan0**:

```
filter          interface      ! lan0
```

7.
Format all events for subsystem **ip**. No other events are formatted. By default, all events are formatted:

```
filter          subsystem      ip   event      *
```

8.
Format all events for subsystem **X25**. No other events are formatted:

```
filter          subsystem      X25   event      *
```

9.
Format only event **5003** for subsystem **ip**. Format all events except **3000** for subsystem **tcp**. No other events are formatted.

```
filter          subsystem       ip      event   5003
filter          subsystem       tcp     event   *,!3000
```

10.
Format only events `5003`, `5004`, `5005`, and `5006` for subsystem `ip`. Format all events except events `3000`, `3002`, and `3003` for subsystem `tcp`. No other events are formatted:

```
filter          subsystem       ip      event   5003-5006
filter          subsystem       tcp     event   *,!3000,!3002-3003
```

11.
Format only those records containing message IDs `9973` and `9974` for subsystem `session` and those not containing message ID `9974` for subsystem `transport`. All records from other subsystems are formatted:

```
#-------------------------------------------------------------
# SUBSYSTEM     REQUEST_TYPE    ARGUMENT1     ARGUMENT2     ...
#-------------------------------------------------------------
ots session     msgid           !*
ots session     msgid           9973
ots session     msgid           9974
ots transport   msgid           !9974
```

12.
Combine LAN, X.25, and general filtering options into one configuration file. Format pduin and pduout data for 15 minutes starting at 3:00 PM on 2 April 1990 for data from the `lan0` interface only.

```
formatter       filter          kind            0x30000000
filter          interface       ! *
filter          interface       lan0
formatter       filter          time_from       15:00:00 04/02/90
formatter       filter          time_through    15:15:00 04/02/90
```

FILES

`/etc/conf/nettlgen`
default subsystem configuration file

`/usr/adm/conslog.opts`
default console logging options filter file

`$HOME/.nettrc` default configuration file for *trace* data if the `-c`*config_file* option is not used on the command line.

`$HOME/.netlogrc`
default configuration file for *log* data if the `-c`*config_file* option is not used on the command line.

SEE ALSO

nettl(1M), nettlconf(1M), nettlgen.conf(4).

AUTHOR

`netfmt` was developed by HP.

NAME

nettl - control network tracing and logging

SYNOPSIS

/etc/nettl -start
/etc/nettl -stop
/etc/nettl -status [*info*]
/etc/nettl -traceon *kind* [*kind ...*] **-entity** *subsystem* [*subsystem ...*] [**-file** *name*] [**-card** *dev_name*] [**-size** *limit*] [**-tracemax** *maxsize*] [**-m** *bytes*]
/etc/nettl -traceoff -entity *subsystem* [*subsystem ...*]
/etc/nettl -log *class* **-entity** *subsystem* [*subsystem ...*]
/etc/nettl -firmlog 0|1|2 -card *dev_name*

DESCRIPTION

nettl is a tool used to capture network events or packets. Logging is a means of capturing network activities such as state changes, errors, and connection establishment. Tracing is used to capture or take a snapshot of inbound and outbound packets going though the network, as well as loopback or header information. A subsystem is a particular network module that can be acted upon such as **NS_LS_DRIVER**, or **X25L2**. **nettl** is used to control the network tracing and logging facility. The command can be used in seven different forms as indicated above. Command forms are discussed first, then options and parameters. Except for the **nettl -status** option, **nettl** can be used only by users who have an effective user ID of 0.

Options

nettl recognizes the following options which can be used only in the combinations indicated above under SYNOPSIS. Options can be specified by spelling out in full or abbreviating as indicated. Argument keywords can be abbreviated as shown. The keywords are case-*in*sensitive.

-start (abbrev: **-st**) Used alone without other options.

Initializes the tracing and logging facility and starts up default logging. Logging is enabled for *all* subsystems as determined by the **/etc/conf/nettlgen.conf** file. Log messages are sent to a log file whose name is determined by adding the suffix **.LOG00** to the log file name specified in the **/etc/conf/nettlgen.conf** configuration file. See *nettlconf*(1M) and *nettlgen.conf*(4) for an explanation of the configuration file. If the log file (with suffix) already exists, it is opened in *append* mode; i.e., new data is added to the file. The name supplied by default is **/usr/adm/nettl** (thus logging starts to file **/usr/adm/nettl.LOG00**). See *Data File Management* below for more information on how the log file is handled.

Note: It is strongly recommended that the tracing and logging facility be started before any other networking. The **/etc/nettl -start** command should be placed in **/etc/netlinkrc** before any other networking commands.

-stop (abbrev: **-sp**) Used alone without other options.

Terminates the trace/log facility. Once this command is issued, the trace/log facility is no longer able to accept the corresponding trace/log calls from the network subsystems.

Note: It is strongly recommended that the tracing and logging facility not be turned off, since information about disasters will be lost. To minimize impact on the system, all subsystems can be set to capture only **disaster**-class log messages.

-status [*info*] (abbrev: **-ss**) Used alone without other options.

Reports tracing and logging facility status. The facility must be operational (i.e. **/etc/nettl -start** has been completed). The default value is **ALL**. *info* defines the type of trace or log information that is to be displayed. *info* can be one of the following:

log	log status information
trace	trace status information

`all` trace & log status information

`-traceon` *kind* [*kind*...]
(abbrev: `-tn`) Requires `-entity` option; `-card` option required for X.25 subsystems only (ignored otherwise); other options recognized but not required.

Starts tracing on the specified subsystem or subsystems. The tracing and logging facility must have been initialized by `nettl -start` for this command to have any effect. The default trace file is standard output, but can be overridden by the optional `-file` argument. If standard output is a tty device, then an informative message is displayed and no trace data is produced.

When tracing is enabled, every operation through the subsystems are recorded if the *kind* mask is matched.

kind defines the trace masks used by the tracing facility before recording a message. *kind* can be entered as one or several of the following keywords or masks:

keyword	mask	keyword	mask
`hdrin`	`0x80000000`	`state`	`0x04000000`
`hdrout`	`0x40000000`	`error`	`0x02000000`
`pduin`	`0x20000000`	`logging`	`0x01000000`
`pduout`	`0x10000000`	`loopback`	`0x00800000`
`proc`	`0x08000000`		

For multiple *kind*s, the masks can be specified separately or combined into a single number. For example, to enable both `pduin` and `pduout` (to trace all packets coming into and out of the node) use `0x30000000`.

`loopback` can be enabled by NS subsystems only. An error may be returned if a given subsystem does not support a particular trace kind. If a `-traceon` is issued on a subsystem already being traced, the tracing mask and optional values are changed to those specified by the new command, but the new `-file`, `-size`, `-tracemax` and `-m` parameters are ignored and a message is issued.

`-entity all`
`-entity` *subsystem* [*subsystem*...]
(abbrev: `-e`) The *subsystem* parameter limits the action (`-status`, `-traceon`, `-traceoff`, or `-log`) to the specified (one or more) protocol layers or software modules.

If `all` is specified, all recognized subsystems are traced except X.25-specific subsystems. To turn on tracing for X.25, use the command

`nettl -tn` *kind* `-e` [*x.25_subsys*] `-c` [*dev_name*]

where the value of [*x.25_subsys*] is `X25L2` or `X25L3`.

The number and names of *subsystem*s on the system is dependent on the products that have been installed. Use the command `nettl -ss all` to obtain a full listing of supported subsystems. *subsystem* examples include:

OSI-examples of subsystems:

`acse_pres`	`ftam_ftp_gw`	`ftp_ftam_gw`
`asn1`	`ftam_init`	`hps`
`cm`	`ftam_resp`	`mms`
`em`	`ftam_vfs`	`ula_utils`
`ots`	`network`	`transport`

LAN-examples of subsystems:

`ns_ls_netisr`	`ns_ls_nfs`	`ns_ls_nft`
`ns_ls_ip`	`ns_ls_ni`	`ns_ls_tcp`
`ns_ls_ipc`	`ns_ls_driver`	`ns_ls_pxp`
`ns_ls_udp`	`ns_ls_loopback`	`ns_ls_x25`

Two X.25-specific subsystems are used for *tracing only*:

`X25L2` `X25L3`

-file *name*
(abbrev: **-f**) Used with the first **-traceon** option only.

The first time the **-traceon** keyword is used, it initializes tracing, creating a file *name* **.TRC0** which receives the binary tracing data. If a trace file of the name *name* **.TRC0** already exists the binary trace data is appended to the end of the file.

To start a fresh trace file, first turn off tracing then turn it back on again using a different *name* (see *Data File Management* below for more information on file naming).

If **-file** is omitted, *binary* trace output goes to standard output. If standard output is a tty device an error message is issued and no tracing is generated.

-card *dev_name*
(abbrev: **-c**) This parameter applies to X.25 subsystems only, for setting up tracing. For other subsystems, this option is ignored and a warning message is issued. Only one X.25 card can be traced at a time.

dev_name specifies a device which corresponds to a network interface card that has been installed and configured. If *dev_name* is not an absolute path name, then **/dev/** is attached in front of *dev_name*. This forms the device file name **/dev/***dev_name*. *dev_name* must refer to a valid X.25 network device file.

-size *limit*
(abbrev: **-s**) Used with first **-traceon** option only.

Sets trace buffer size (in Kbytes) used to hold trace messages until they are written to the file. Default value for this buffer is 32 Kbytes. The possible range for this parameter is 1 through 512 Kbytes. Setting this value too low increases the possibility of dropped trace messages from kernel subsystems.

-tracemax *maxsize*
(abbrev: **-tm**) Used with first **-traceon** option only.

Tracing uses a circular file method such that when one file fills up, a second is used. Two trace files can exist on a system at any given time. See *Data File Management* below for more information on file behavior.

maxsize specifies the maximum size of both trace files combined. Specify *maxsize* in multiples of 1 Kbyte. If this option is not specified, a default size of 1 Mbytes is used. *maxsize* can range in value from 100 through 99999 Kbytes.

-m *bytes*
Used with the first **-traceon** option only. Number of bytes to trace. This option allows the user to specify the number of bytes to be captured in the trace packet. The user may prefer not to capture an entire PDU trace, such as when interested only in the header. *bytes* is the number of bytes traced. Default: the entire packet is traced. This option is currently recognized for tracing by the following subsystems only:

```
ns_ls_driver      ns_ls_ni
x2512             x2513
```

-traceoff
(abbrev: **-tf**) Requires **-entity** option.

Disables tracing of *subsystem*s specified by the **-entity** option. If **ALL** is specified as an argument to the **-entity** option, all tracing is disabled. The trace file remains, and can be formatted by using the **netfmt** command to view the trace messages it contains (see *netfmt*(1M)).

-log *class*
(abbrev: **-l**) Requires **-entity** option.

Controls the class of log messages that are enabled for the subsystems specified by the **-entity** option.

class specifies the logging class. Available classes are:

Full	Abbrev	Mask
informative	i	1
warning	w	2

```
error           e       4
disaster        d       8
```

Classes can be specified as keywords or as a numeric mask depicting which classes to log. If you choose to indicate several classes at once, be sure to separate each log class with a space.

disaster logging is always on. The default logging classes for each subsystem can be configured into a configuration file, **/etc/conf/nettlgen.conf**. When the tracing/logging facility is started, the information in the configuration file is read and subsystems are enabled for logging with the specified classes. To change the log class, use the **nettl -log** *class* **-entity** *subsystem* command with a new log class value. The **nettl -log** *class* **-entity** *subsystem* command can be run for different log classes and different entities if desired.

-firmlog 0|1|2
(abbrev: **-fm**) Requires **-card** option. S800, X.25 only.

Sets the X.25/800 interface card logging mask to level 0, 1, or 2. The default level is 0. The X.25/800 interface logs a standard set of messages. A level of 1 specifies cautionary messages as well as the default messages. A level of 2, specifies information messages in addition to cautionary and default messages. This option is recognized only by the **NS_LS_X25** subsystem.

Data File Management

Data files created by the tracing and logging facility require special handling by the facility that the user must be aware of. When files are created, they have the suffix **.LOG00** or **.TRC0** appended to them, depending on whether they are log or trace files, respectively. This facility is used to keep the files distinct for cases where the user specifies the same name in both places. Also, the files implement a type of circular buffer, with new data always going into the file appended with **.LOG00** or **.TRC0**. When the files are full, they are renamed to the next higher number in their sequence; i.e., **.LOG01** or **.TRC1** and new files with the *0* extension are created. Currently only two generations of files are possible; thus only two log files appear on the system simultaneously (**.LOG00** and **.LOG01**.) The same is true for trace files as well; only two trace files exist with the same file name, not counting the postfix of **.TRC0** and **.TRC1**.

> **Note:** The file name prefix specified by the user must not exceed eight characters so that the file name plus suffix does not exceed fourteen characters. Longer names are truncated silently. To see the actual name of the trace or log file, use the **/etc/nettl -status all** command.

Console Logging

Console logging is controlled by the configuration information in the **/etc/conf/nettlgen.conf** and **/usr/adm/conslog.opts** files by default. All log messages written to the console as a result of this configuration information are in a special short form. If more information is desired on the console, the **netfmt** formatter can be used to direct output to the console device. This may be most useful in an X windows environment. See examples below on how to do this.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported in data; single-byte character code sets are supported in filenames.

EXAMPLES

1. Initialize the tracing/logging facility:

```
nettl -start
```

2. Change log class to **warning** for all the subsystems. **disaster** logging is always on for all subsystems.

```
nettl -log w -e all
```

3. Turn on inbound PDU tracing for the subsystems **ula_utils**, **hps** (all trace kinds are enabled by ORing the masks) and send binary trace messages to file **/usr/adm/trace.TRC0**.

```
nettl -traceon pduin -entity ula_utils hps -file /usr/adm/trace
```

4. Turn on outbound PDU tracing for X.25 level two, and subsystem **NI**. Trace messages go to the trace file

set up in the previous example. This example also uses the abbreviated options. Tracing for X.25 requires a `-card` parameter to indicate which X.25 card to trace. If you choose not to trace X.25 by omitting `X25L2`, the card parameter is ignored.

```
nettl -tn  pduout -e X25L2 ns_ls_ni -c x25_0
```

5.
Determine status of tracing from example 3.

```
nettl -status trace
```

The resulting information should resemble the following:

```
Tracing Information:

Trace Filename:             /usr/adm/trace.TRCx
User's ID:          0       Buffer Size:            32768
Messages Dropped:   0       Messages Queued:        0

Subsystem Name:             Trace Mask:
ULA_UTILS                       20000000
HPS                             20000000
X25L2                           1000000
NS_LS_NI                        1000000
```

6.
Enable `pdu` tracing for `lan` subsystems. Binary trace data goes to file `/usr/adm/LAN.TRC0`.

The `-file` option of this command is only valid the first time tracing is called. To change the trace output file, stop tracing and start up again. The trace file is not automatically reset with the `-file` option. This example assumes that the `-traceon` option is being used for the first time.

```
nettl -tn pduin pduout -e ns_ls_driver -file /usr/adm/LAN
```

7.
Terminate the tracing and logging facility.
Note: It is *strongly* recommended that the tracing and logging facility be turned on before any networking is started and remain on as long as networking is being used.

```
nettl -stop
```

WARNINGS

Tracing or logging to a file may not be able to keep up with a busy system, especially when extensive tracing information is being gathered. If some data loss is encountered, the trace buffer size can be increased. Be selective about the number of subsystems being traced, as well as the log class messages being captured.

The `nettl` and `netfmt` commands read the `/etc/conf/nettlgen.conf` file each time they are run (see *nettl*(1M) and *netfmt*(1M)). If the file becomes corrupted, these commands will no longer be operational.

FILES

`/etc/conf/nettlgen.conf`	tracing and logging subsystem configuration file.
`/usr/adm/conslog.opts`	default console logging options filter file as specified in `/etc/conf/nettlgen.conf`.
`/usr/adm/nettl.LOG00`	default Log file as specified in `/etc/conf/nettlgen.conf`.
`/dev/nettrace`	kernel trace pseudo-device file.
`/dev/netlog`	kernel log pseudo-device file.

AUTHOR

`nettl` was developed by HP.

SEE ALSO

netfmt(1M), nettlconf(1M), nettlgen.conf(4).

NAME

nettlconf - configure network tracing and logging command subsystem database

SYNOPSIS

/etc/conf/nettlconf -id *ssid* **-name** *ssname* [**-class** *logclass*] [**-kernel**] **-lib** *sslib* **-msg** *ssmsgcat* [**-fmtfn** *fmtfunc*] [**-optfn** *optfunc*] **-group** *ssgrpname* [**-umedia** *media*]

/etc/conf/nettlconf -delete *ssid*

DESCRIPTION

nettlconf maintains the database file **/etc/conf/nettlgen.conf** which contains information necessary for each subsystem that uses network tracing and logging. This database contains one entry for each subsystem. If a subsystem already exists with the same ID, the values given are substituted for those in the database; otherwise a new entry is created. The first form of the command shown above adds or updates a new subsystem to the database. The second form, **nettlconf -d** *ssid*, removes a subsystem entry from the database.

nettlconf is used primarily during product installation or product update. The **nettlconf** command should be executed from within the customize script at the end of an install or update. The customize script should execute the **nettlconf** command for each subsystem it installs.

Options

-id *ssid* (abbrev: **-i**) *ssid* (subsystem ID number) is used as the key field in the **nettlgen.conf** database. This is a required field.

-name *ssname* (abbrev: **-n**) *ssname* is the subsystem-name mnemonic. This string is used to identify the subsystem on the **nettl** command line and also in the header output from the formatter (see *nettl*(1M)). This is a required field.

-class *logclass* (abbrev: **-c**) *logclass* is the default log class mask assigned to the subsystem at start-up of the tracing/logging facility. For multiple classes, the masks can be combined into a single number. For example, to initially log **DISASTER** and **ERROR** events use **12** as the *logclass*. Default is an empty field in **nettlgen.conf**. **nettl** substitutes **8** (disaster) for an empty class field. This is an optional field.

Class	Abbreviation
informative	1
warning	2
error	4
disaster	8

-kernel
(abbrev: **-k**) flags the given subsystem as a kernel subsystem. **nettl** uses this information to control certain tracing and logging properties of the subsystem. A subsystem is defaulted to non-kernel unless this option is used. This is an optional field.

-lib *sslib*
(abbrev: **-l**) *sslib* is the name of the shared library where the subsystem formatter resides. This should be an absolute path name unless the library resides in **/usr/lib**. Multiple subsystems can reference the same library. This is a required field.

-msg *ssmsgcat*
(abbrev: **-m**) *msg* is the name of the message catalog *excluding* the pathname and **.cat** filename extension. Multiple subsystems can refer to the same message catalog. This is a required field.

-fmtfn *fmtfunc*
(abbrev: **-f**) *fmtfunc* specifies the function to call when formatting data from the given subsystem. Default is to form the function name from the subsystem ID as follows:

subsys_*N*_format

where *N* is the subsystem ID number. If a null function is needed for this subsystem, specify

-f NULL

This is an optional field.

-optfn *optfunc*
(abbrev: **-o**) *optfunc* specifies the function used to process options in the **netfmt** filter configuration file (see *netfmt*(1M)). The default is an empty field in **nettlgen.conf**. **netfmt** assumes a NULL function for an empty *optfunc* field. This is an optional field.

-group *ssgrpname*
(abbrev: **-g**) *ssgrpname* is a group name associated with the subsystem. Several subsystems can be grouped together so that a common banner is printed in the formatted header. This is a required field.

-umedia *media*
(abbrev: **-u**) *media* is the update media information that describes the context of the product installation. This information is necessary for proper product installation in clustered environments. *media* is passed as a parameter to the product customize script by **update** (see *update*(1M)). The product customize script should use this option when *media* has a value. This is an optional field.

-delete *ssid*
(abbrev: **-d**) Deletes the *ssid* (subsystem ID) from the database.

WARNINGS

The **nettlconf** utility is intended primarily for use by HP subsystems to configure themselves into the tracing and logging facility at installation time. System administrators may wish to use this command to alter the default logging class each subsystem starts up with, but no other information about the subsystem should be changed. Only an editor such as **vi** should be used to change the default logging information kept in the **/etc/conf/nettlgen.conf** file.

The **nettl** and **netfmt** commands read the **/etc/conf/nettlgen.conf** file each time they are executed. If the file becomes corrupted these commands cannot function.

AUTHOR

nettlconf was developed by HP.

FILES

/etc/conf/nettlgen.conf subsystem configuration file maintained by **nettlconf**

SEE ALSO

netfmt(1M), nettl(1M), update(1M), nettlgen.conf(4).

NAME

nettlgen - generate network tracing and logging commands

SYNOPSIS

`/etc/conf/nettlgen`

DESCRIPTION

`nettlgen` is obsolete. The functionality that it provided is no longer necessary.

SEE ALSO

nettl(1M), netfmt(1M), nettlconf(1M), nettlgen.conf(4).

NAME

newfs - construct a new file system

SYNOPSIS

File Name Length Same as Root

/etc/newfs [**-F**] [**-n**] [**-v**] [*mkfs-options*] *special disk_type*

Short File Name System

/etc/newfs -S [**-F**] [**-n**] [**-v**] [*mkfs-options*] *special disk_type*

Long File Name System

/etc/newfs -L [**-F**] [**-n**] [**-v**] [*mkfs-options*] *special disk_type*

DESCRIPTION

newfs is a "friendly" front-end to the **mkfs** program (see *mkfs*(1M)). **newfs** looks in disk description file **/etc/disktab** for the type of disk a file system is being created on, calculates the appropriate parameters to use in calling **mkfs**, then builds the file system by forking **mkfs** and, if the file system is a root section, installs the necessary bootstrap programs in the initial 8192 bytes of the device.

newfs creates the file system with a rotational delay value (see *tunefs*(1M)) as based on the interface (SCSI, HP-FL, HPIB) and other characteristics of the disk drive. To get the appropriate rotational delay value, turn immediate reporting on or off before rather than after creating the file system.

Options

newfs recognizes the following command-line options and arguments:

-L or **-S** — There are two types of HFS file systems. They are distinguished mainly by differing directory formats that place different limits on the length of directory entries (file names). By default, **newfs** creates a file system of the same type as the root file system. However, the type of file system can be explicitly specified by using the **-L** or **-S** option. **-L** (long file names) creates a file system that allows file names up to **MAXNAMLEN** (255) bytes long; **-S** (short file names) creates a file system that allows file names not more than **DIRSIZ** (14) bytes long.

-F — Forces **newfs** to continue processing on a mounted file system. If **-F** is not specified, **newfs** prompts the user and waits for a reply. Note that **newfs** does not work on a swap device, even if **-F** is specified.

-n — Prevents bootstrap programs from being installed.

-v — (verbose) **newfs** prints out its actions, including the parameters passed to **mkfs**.

special — character (raw) special file for the disk.

disk_type — Type of disk as specified in **/etc/disktab**.

mkfs Default-Override Options

The following additional command-line options can be used to override default parameters passed to **mkfs**:

-s *size* — File system size in **DEV_BSIZE** blocks (defined in <**sys/param.h**>).

-b *block-size* — File system block size in bytes.

-f *frag-size* — File system fragment size in bytes.

-t *tracks_per_cylinder* — Number of tracks per cylinder.

-c *#cylinders_per_group* — Number of cylinders per cylinder group in a file system. The default value used is 16.

-m *free_space_percent* — Percentage of space reserved from normal users; the minimum free space threshold. Default value is 10 percent.

-r *revolutions_per_minute* — Disk speed in revolutions per minute (normally 3600).

-i number_of_bytes_per_inode — Specifies the density of inodes in the file system. Default is to create an inode for each

2048 bytes of data space. If fewer inodes are desired, a larger number should be used; to create more inodes a smaller number should be given.

Access Control Lists (ACLs)

Every file with one or more optional ACL entries consumes an extra (continuation) inode. If you anticipate significant use of ACLs on a new file system, you can allocate more inodes by reducing the value of the argument to the `-i` option appropriately. The small default value typically causes allocation of many more inodes than are actually necessary, even with ACLs. To evaluate the need for extra inodes, run `bdf -i` on existing file systems. For more information on access control lists, see *acl*(5).

Note that access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

FILES

`/etc/disktab` disk geometry and file system section information

DEPENDENCIES

Series 700/800

`newfs` does not install bootstrap programs in a root section of a Series 800 system because the boot programs are kept in a separate section. On Series 700 systems, `newfs` does not install the bootstrap programs, in order to conserve disk space. On both Series 700 and Series 800 systems, the `mkboot` command can be used to install the bootstrap programs.

AUTHOR

`newfs` was developed by the University of California, Berkeley and HP.

SEE ALSO

bdf(1M), fsck(1M), mkboot(1M), mkfs(1M), tunefs(1M), disktab(4), fs(4), acl(5).

NAME

nfsd, biod - NFS daemons

SYNOPSIS

`/etc/nfsd` [*nservers*]

`/etc/biod` [*nservers*]

DESCRIPTION

`nfsd` starts the NFS server daemons that handle client filesystem requests (see *nfs*(7)). *nservers* is the number of file system request daemons that start. This number should be determined by the load expected on the server system. To obtain the best performance in most cases, set *nservers* to four.

`biod` starts *nservers* asynchronous block I/O daemons. This command is used on an NFS client to buffer cache handle read-ahead and write-behind. *nservers* is a number greater than zero. For best performance, set *nservers* to four.

AUTHOR

`nfsd` was developed by Sun Microsystems, Inc.

SEE ALSO

mountd(1M), exports(4).

INTERNATIONAL SUPPORT

messages

NAME

nfsstat - Network File System statistics

SYNOPSIS

/usr/etc/nfsstat [**-csnrz**] [*namelist*]

DESCRIPTION

nfsstat displays statistical information about the Network File System (NFS) and Remote Procedure Call (RPC) interfaces to the kernel. It can also reinitialize this information. If no options are given the default is equivalent to

```
nfsstat -csnr /hp-ux
```

which prints everything and reinitializes nothing.

Options

nfsstat recognizes the following options and command-line arguments:

-c Display client information. **nfsstat** displays only the client side NFS and RPC information. Combine **-c** with the **-n** and **-r** options to print client NFS or client RPC information only.

Client NFS statistics include the total number of RPC calls made on behalf of NFS (**calls**), the number of RPC calls that failed (**badcalls**), the number of times the kernel attempted to get a client structure (**nclget**), the number of times the kernel waited to get a client structure (**nclsleep**), and percentages for different file system operations.

Client RPC statistics include the total number of RPC calls (**calls**), the number of RPC calls that failed (**badcalls**), the number of RPC retransmissions required (**retrans**), the number of RPC packets received that had an out of date transaction ID (**badxid**), the number of RPC packets sent that did not receive a response within the timeout period (**timeout**), and two fields reserved for future use (**wait** and **newcred**).

-s Display server information. Works like the **-c** option above.

Server NFS statistics include the total number of RPC calls made on behalf of NFS (**calls**), the number of RPC calls that failed (**badcalls**) and percentages for different file system operations.

Server RPC statistics include the total number of RPC calls (**calls**), the number of RPC calls that failed (**badcalls**), the number of empty RPC packets received (**nullrecv**), the number of RPC packets received that were too short to be valid packets (**badlen**), and the number of badly formatted RPC packets received (**xdrcall**).

-n Display NFS information. The NFS information is kernel information only. **nfsstat** displays NFS information for both the client and server side. Combine **-n** with the **-c** and **-s** options to print client or server NFS information only.

-r Display RPC information. The RPC information is kernel RPC information only. Works like the above **-n** option.

-z Print the current statistics then zero (reinitialize) them. Combine **-z** with any of the options to zero particular sets of statistics after printing them. The user must have write permission on **/dev/kmem** for this option to work.

namelist This argument is taken as an alternate file name for use instead of **/hp-ux** in obtaining the system *namelist*.

AUTHOR

nfsstat was developed by Sun Microsystems, Inc.

FILES

/hp-ux	system namelist
/dev/kmem	kernel memory

INTERNATIONAL SUPPORT

messages

NAME

nrglbd - Non-Replicatable Global Location Broker daemon

SYNOPSIS

/etc/ncs/nrglbd [*-version*]

DESCRIPTION

The Global Location Broker (GLB), part of the Network Computing System (NCS), helps clients to locate servers on a network or internet. The GLB database stores the locations (that is, the network addresses and port numbers) where server processes are running. A daemon maintains this database and provides access to it.

There are two versions of the GLB daemon: **glbd** and **nrglbd**. Only the replicatable version, **glbd**, is provided for Domain/OS, HP-UX, SunOS, and ULTRIX systems. For other systems, the non-replicatable version, **nrglbd**, is provided. The two versions of the daemon should not coexist on a network. (For HP-UX systems, which may have both **glbd** and **nrglbd**, use of only **glbd** is strongly recommended.)

This entry describes only **nrglbd**.

Typically, **nrglbd** is started in the background at boot time. Unless the host is an MS-DOS system, a Local Location Broker daemon (**llbd**) must be running on the local host when **nrglbd** is started.

On MS-DOS systems that use DDS network protocols, **nrglbd** uses DDS protocols; on other systems, it uses IP protocols.

Do not run more than one **nrglbd** on a network or internet, and do not run an **nrglbd** and a **glbd** on the same network or internet. On HP-UX systems, do not run **nrglbd** at all; run **glbd** instead.

See *Managing NCS Software* for more information about Location Broker configuration.

Options

-version

Display the version of NCK that this **nrglbd** belongs to, but do not start the daemon.

SEE ALSO

glbd(1M), lb_admin(1M), llbd(1M).

Managing NCS Software.

NAME

ocd - outbound connection daemon used by DDFA software

SYNOPSIS

ocd -n*node_name* **-f***pseudonym* [**-b***board_no*] [**-p***port_no*] [**-c***config_file*]

DESCRIPTION

The **Outbound Connection Daemon** (**ocd**) is part of the **HP Datacommunications and Terminal Controller** (DTC) **Device File Access** (DDFA) software. It manages the connection and data transfer to the remote DTC port. It can be spawned from the **Dedicated Port Parser** (**dpp**) or run directly from the shell.

For performance reasons, **ocd** does not have a debug mode; however, a version called **ocdebug** with debug facilities is available (see *ocdebug*(1m) for more information.

See *ddfa*(7) for more information on how to configure and install the DDFA software, and for an explanation of how it works.

ocd recognizes the following options and command-line arguments:

-n*node_name* Mandatory. This is the IP address of the terminal server or the port.

-f*pseudonym* Mandatory. This is the absolute or relative path to the device file which is linked by the software to the reserved pty. Applications use the pseudonym, not the dynamically allocated pty slave.

-b*board_no* Optional. This refers to the board number of the DTC. If it is omitted, the port number option must contain the full TCP service port address. **-b** and **-p** must not be used if the IP address given in **-n** is the IP address of a port.

If the **-n** option explicitly names a DTC port, the **-b** option is not needed.

-p*port_no* Optional. This is the DTC port number. If the **-b** option is omitted, the port number must be the TCP service port address that will be used by the software to access the port. If the value is omitted, the value **23** (Telnet) is used by default.

-c*config_file* Optional. This is the name (including the absolute path) of the configuration file used to profile the DTC port. If this value is omitted, the default values specified in the default **pcf** file (**/etc/newconfig/ddfa/pcf**) are used. If the **pcf** does not exist, an error messageis be logged, and the following values are used (note that the values for *open_tries* and *open_timer* are different from the default values):

```
telnet_mode:      enable
timing_mark:      enable
telnet_timer:     120
binary_mode:      disable
open_tries:       0
open_timer:       0
close_timer:      0
status_request:   disable
status_timer:     30
eight_bit:        disable
tcp_nodelay:      enable
```

ocd logs important messages and error conditions to **/usr/adm/syslog**.

FILES

/etc/dpp
/etc/ocdbug
/etc/ocd
/etc/dpp_login.bin
/etc/utmp.dfa
/etc/newconfig/ddfa/pcf
/etc/newconfig/ddfa/dp

SEE ALSO
ddfa(7) dp(4) dpp(1m) ocdebug(1m) pcf(4).

NAME

ocdebug - Outbound Connection Daemon debug utility (used by DDFA software)

SYNOPSIS

ocdebug -n *node_name* **-f** *pseudonym* [**-b** *board_no*] [**-p** *port_no*] [**-c** *config_file*] [**-d** *level*]

DESCRIPTION

The Outbound Connection Daemon (**ocd**) is part of the HP Datacommunication and Terminal Controller (DTC) Device File Access (DDFA) software. It manages the connection and data transfer to the remote DTC port. For performance reasons, it does not have a debug mode; **ocdebug** is a version of **ocd** with debug facilities. See *ocd*(1M) for more information about **ocd**.

See *ddfa*(7) for more information about how to configure and install the DDFA software, and for an explanation of how it works.

Debugging may be toggled interactively by sending the **SIGUSR1** signal to the process by typing **kill -16** *pid*.

Options and Command-Line Arguments

ocdebug recognizes the following arguments (note that apart from **-d** they are the same as the **ocd** command-line arguments):

-n *node_name* Mandatory. This is the IP address of the server or the port.

-f *pseudonym* Mandatory. This is the absolute or relative path to the device file that is linked by the software to the reserved pty. Applications use the pseudonym; not the dynamically allocated pty slave.

-b *board_no* Optional. This refers to the board number of the DTC. If it is omitted, the port number option must contain the full TCP service port address.

If the **-n** option explicitly names a DTC port, the **-b** option is not needed.

-p *port_no* Optional. This is the DTC port number. If the **-b** option is omitted, the port number must be the TCP service port address used by the server to access the port. If *port_no* is not specified, a value of **23** (Telnet) is used by default.

-c *config_file* Optional. This is the path to the Port Configuration File (**pcf**) used for profiling the DTC port. If this value is omitted, the default values specified in the default **pcf** file (**/etc/newconfig/ddfa/pcf**) is used. If the **pcf** doesn't exist, an error message is logged, and the following values are used (note that the values for *open_tries* and *open_timer* are different from the default values):

telnet_mode: enable
timing_mark: enable
telnet_timer: 120
binary_mode: disable
open_tries: 0
open_timer: 0
close_timer: 0
status_request: disable
status_timer: 30
eight_bit: disable
tcp_nodelay: enable

-d *level*

Optional. This indicates the level of debugging. Levels can be added together to accumulate dubugging functions. For example, **-d7** enables all levels, and **-d3** enables only the first two levels. The levels are:

0 No debug messages.
1 Trace procedure entry/exit logged.
2 Additional tracking messages logged.
4 Data structures dumped.

Debug messages are logged to the file **/usr/adm/ocd***pid*, and the file name is displayed at the start of debugging.

FILES

```
/etc/dpp
/etc/ocdbug
/etc/ocd
/etc/dpp_login.bin
/etc/utmp.dfa
/etc/newconfig/ddfa/pcf
/etc/newconfig/ddfa/dp
```

SEE ALSO

ddfa(7), dp(4), dpp(1M), ocd(1M), pcf(4).

NAME

opx25 - execute HALGOL programs

SYNOPSIS

/usr/lib/uucp/X25/opx25 [**-f***scriptname*] [**-c***char*] [**-o***file-descriptor*] [**-i***file-descriptor*] [**-n***string*] [**-d**] [**-v**]

DESCRIPTION

HALGOL is a simple language for communicating with devices such as modems and X.25 PADs. It has simple statements like 'send xxx' and 'expect yyy' that are described below.

Options:

opx25 recognizes the following options:

-f script Causes *opx25* to read script as the input program. If **-f** is not specified then *opx25* reads stdin for the script.

-c char Causes *opx25* to use 'char' as the first character in the input stream instead of actually reading it from the input descriptor. This is useful sometimes when the program that calls *opx25* is forced to read a character but then cannot "unread" it.

-o number Causes *opx25* to use 'number' for the output file descriptor (ie, the device to use for 'send'). The default is 1.

-i number Causes *opx25* to use 'number' for the input file descriptor (ie, the device to use for 'expect'). The default is 0.

-n string Causes *opx25* to save this string for use when "\#" is encountered in a "send" command.

-d Causes *opx25* to turn on debugging mode.

-v Causes *opx25* to turn on verbose mode.

An *opx25* script file contains lines of the following type:

(empty) Empty lines are ignored.

/ Lines beginning with a slash "/" are ignored (comments)

ID ID denotes a label. ID is limited to alphanumerics or "_".

send STRING STRING must be surrounded by double quotes. The text is sent to the device specified by the **-o** option.
Non-printable characters are represented as in C; that is, as \DDD, where DDD is the octal ascii character code. "\#" in a send string is the string that followed the -n option.

break Send a break "character" to the device.

expect NUMBER STRING
Here NUMBER is how many seconds to wait before giving up. 0 means wait forever, but this isn't advised. Whenever STRING appears in the input within the time allotted, the command succeeds. Thus, it isn't necessary to specify the entire string. For example, if you know that the PAD will send several lines followed by an "@" prompt, you could just use "@" as the string.

run program args
The program (sleep, date, whatever) is run with the args specified. Don't use quotes here. Also, the program is invoked directly (with execp), so wild cards, redirection, etc. are not possible.

error ID If the most recent expect or run encountered an error, go to the label ID.

exec program args
Like run, but doesn't fork.

echo STRING Like send, but goes to stderr instead of to the device.

set debug Sets the program in debug mode. It echoes each line to **/tmp/opx25.log,** as well as giving the result of each expect and run. This can be useful for writing new scripts.

The command "set nodebug" will turn off this feature.

set log Sends subsequent incoming characters to **/usr/spool/uucp/.Log/X25LOG**. This can be used in the *.in file as a security measure, since part of the incoming data stream contains the number of the caller. There is a similar feature in *getx25;* it writes the time and the login name into the same logfile. The command "set nolog" will turn off this feature.

set numlog Like "set log", only better in some cases, since it sends only digits to the log file, and not other characters. The command "set nonumlog" will turn off this feature.

timeout NUMBER
Sets a global timeout value. Each expect uses time in the timeout reservoir; when this time is gone, the program gives up (exit 1). If this command isn't used, there is no global timeout. Also, the global timeout can be reset any time, and a value of 0 turns it off.

exit NUMBER Exits with this value. 0 is success, anything else is failure.

You can test configuration files, sort of, by running *opx25* by hand, using the argument "-f" followed by the name of the script file. The program in this case sends to, and expects from, standard output and input, so you can type the input, observe the output, and see messages with the **echo** command. See the file **/usr/lib/uucp/X25/ventel.out** for a good example of Halgol programming.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

AUTHOR

opx25 was developed by HP.

SEE ALSO

getx25(1), uucp(1).

UUCP tutorial in *Remote Access Users Guide* .

NAME

pcnfsd - PC-NFS daemon

SYNOPSIS

/etc/pcnfsd [*spooldir*] [**-l** *log_file*]

DESCRIPTION

pcnfsd is called by Personal Computer NFS (PC-NFS) users to perform PC user authentication on HP-UX servers. This allows a PC user to access NFS file systems with the appropriate user ID and group ID. It also allows access to HP-UX printer facilities. Refer to PC-NFS documentation for details on the use of the PC-NFS product.

OPTIONS

pcnfsd recognizes the following options and command-line arguments:

spooldir Use the named directory as the PC-NFS spool directory. This directory is used by **pcnfsd** to hold files for printing. The default directory is **/usr/tmp**.

-l *log_file* Log any errors to the named *log_file*. **pcnfsd** writes its messages directly to system console **/dev/console** if **pcnfsd** is started without the **-l** (ell) option.

Information logged to the file includes:

- Date and time of the error,
- Host name, process id, and name of the function generating the error, and
- Error message.

Note that different services can share a single log file because enough information is included to uniquely identify each error.

AUTHOR

pcnfsd was developed by Sun Microsystems, Inc. and HP.

SEE ALSO

lp(1), nfsd(1M).

INTERNATIONAL SUPPORT

messages

NAME

pcserver - Basic Serial and HP AdvanceLink server

SYNOPSIS

pcserver [**-n**] [**-l** [*log_file*]]

DESCRIPTION

pcserver is the hostside server program for Basic Serial and AdvanceLink. *pcserver* is both started and terminated by an application program on a PC.

pcserver supports both the Basic Serial and the AdvanceLink protocols. Basic Serial is a library of routines that support various types of 'services' between a PC and a serially connected host computer. These include file transfer and remote interprocess communication.

AdvanceLink is a terminal emulation program that also supports file transfers between a PC and host system over various physical connections.

The following options are recognized by *pcserver*:

-l [*logfile*] Enable packet logging. This should only be used when needed for debugging. If *logfile* is not specified, the file **s-log** is used in the default logging directory. The default logging directory is defined in file **server.pro**. *pcserver* looks for a version of **server.pro** in the user's home directory, and if none is found uses the system defaults in **/usr/adm/server.pro**. If **logfile** exists, logging is appended to it. If the file does not exist, it is created.

-n Use some form of "netmode" for data encryption. Netmode should be used in situations where certain characters might create undesired results (such as when a PAD is being used and device control characters must masked). This option only tells *pcserver* that some form of netmode will be used. The actual details are negotiated between *pcserver* and the PC application. For a more comprehensive discussion on netmode, see *Using Basic Serial Connection Files*.

pcserver is designed to be invoked by a PC application program rather than from the command line. In order for the connection to be correctly established, the PC and host port must be properly configured.

If you are using *pcserver* to manage a session between a PC and a hostside application (via Basic Serial), you may need to use a Basic Serial connection file to actually log in to your account. Establishing connections using Basic Serial connection files is a sensitive operation. Before attempting to use them, you should read the manual *Using Basic Serial Connection Files*.

If you are using *pcserver* to transfer files between a PC and a host machine via Advancelink, use the following AdvanceLink commands:

&HOSTCOPY "pcserver"
&TERMINATOR "$"

If your prompt does not end with **$**, replace the **$** in the terminator command with the last character in your normal prompt.

To permanently configure AdvanceLink for the HP-UX version of *pcserver*, refer to the *Using AdvanceLink* manual for more information.

NOTES

If your screen displays a "Command not found" message when you choose START TRANSFER from AdvLink, either *pcserver* has not yet been installed on your HP-UX system, or it has been installed in a directory that is not part of your current path.

HP-UX treats files containing binary or ASCII data identically. Therefore it is up to the user to specify the desired file type when using *pcserver* to transfer files with Advancelink. The difference between the two is that during ASCII transfers, *pcserver* maps HP-UX line-feed characters to the MS-DOS carriage-return/line-feed pair. This produces incorrect results when transferring a binary file as an ASCII file.

Also, older versions of AdvanceLink show totally inaccurate estimates for file transfer times. This does not interfere with the actual transfer.

If the PC is reset while a transfer is taking place, it may temporarily appear to be a "dead" terminal port. This is no cause for alarm; left to its own devices, *pcserver* will restore the port in a short time. In the worst

case, it could take six timeout periods (6 * 20 = 120 seconds). For faster response, press the Break key a few times to terminate *pcserver* immediately.

FILES

/usr/bin/pcserver	the executable program
/usr/adm/server.pro	system-wide logging profile
$HOME/server.pro	local logging profile

SEE ALSO

Using AdvanceLink
Describes protocol and how to use AdvanceLink.

Using Basic Serial Connection Files
Describes Basic Serial and how connection files should be used.

NAME

pdc - processor-dependent code (firmware)

DESCRIPTION

pdc is the firmware that implements all processor-dependent functionality, including initialization and self-test of the processor. Upon completion, it loads and transfers control to the initial system loader (*isl*(1M)). Firmware behavior varies somewhat, depending on the hardware series as described below.

Series 800 Behavior

To load *isl* from an external medium, *pdc* must know the particular device on which *isl* resides. Typically the device is identified by the Primary Boot Path that is maintained by *pdc* in Stable Storage. A *path* specification is a series of decimal numbers each suffixed by '/', indicating bus converters, followed by a series of decimal numbers separated by '.', indicating the various card and slot numbers and addresses. The first number, not specifying a bus converter, is the MID-BUS module number (that is, slot number times four) and followed by the CIO slot number. If the CIO slot contains an HP-IB card, the next number is the HP-IB address, followed by the unit number of the device if the device supports units. If the CIO slot contains a terminal card, the next number is the port number, which must be zero for the console.

When the processor is reset after initialization and self-test complete, *pdc* reads the Console Path from Stable Storage, and attempts to initialize the console device. If the initialization fails, *pdc* attempts to find and initialize a console device. Algorithms used to find a console device are model-dependent. *pdc* then announces the Primary Boot, Alternate Boot, and Console Paths.

If *autoboot* (see *isl*(1M)) is enabled, *pdc* provides a 10-second delay, during which time the operator can override the *autoboot* sequence by typing any character on the console. If the operator does not interrupt this process, *pdc* initializes and reads *isl* from the Primary Boot Path. On models that support autosearch, if this path is not valid and *autosearch* (see *isl*(1M)) is enabled, *pdc* then searches through the MID-BUS modules and CIO slots to find a bootable medium. Currently, autosearch is only implemented on the model 825.

If the *autoboot* sequence is unsuccessful, overridden by the operator, or not enabled in the first place, *pdc* interactively prompts the operator for the Boot Path to use. Any required path components that are not supplied default to zero.

The Primary Boot, Alternate Boot, and Console Paths as well as *autoboot* and *autosearch* enable can be modified via *isl*.

Series 700 Behavior

To load *isl* from an external medium, *pdc* must know the particular device on which *isl* resides. Typically the device is identified by the Primary Boot Path that is maintained by *pdc* in Stable Storage. A *path* specification is an I/O subsystem mnemonic that varies according to hardware model.

When the processor is reset after initialization and self-test complete, *pdc* reads the Console Path from Stable Storage, and attempts to initialize the console device. If the initialization fails, *pdc* attempts to find and initialize a console device. Algorithms used to find a console device vary according to hardware model.

If *autoboot* and *autosearch* (see *isl*(1M)) are enabled, *pdc* waits for approximately 10 seconds during which time the operator can override the *autoboot* sequence pressing and holding the ESC (escape) key on the console.

The system then begins a search for potentially bootable devices. If allowed to complete, a list of potentially bootable devices is displayed, labeled with abbreviated path identifiers (P0, P1, etc). A simple menu is then displayed where the user can:

- Boot a specific device, using the abbreviated path identifier, or the full mnenomic.
- Start a device search where the contents are searched for IPL images (note the first search only identified devices and did not check the contents).
- Enter the boot administration level.
- Exit the menu and return to autobooting
- Get help on choices

The search of potentially bootable devices can be aborted by pressing and holding the escape key. The search for device contents can also be aborted by pressing and holding the escape key.

If the operator does not interrupt the search process, *pdc* initializes and reads *isl* from the Primary Boot Path.

If the *autoboot* sequence is unsuccessful, overridden by the operator, or not enabled in the first place, *pdc* executes the device search and enters the menu described above.

The Primary Boot, Alternate Boot, and Console Paths as well as *autoboot* and *autosearch* enable can be modified via *isl* or at the pdc boot administration level.

SEE ALSO

boot(1M), hpuxboot(1M), isl(1M).

NAME

pdfck - compare Product Description File to File System

SYNOPSIS

pdfck [**-n**] *alternate_root*] *PDF*

DESCRIPTION

pdfck is a program that compares the file descriptions in a PDF (Product Description File) to the actual files on the file system. It is intended as a tool to audit the file system and detect corruption and/or tampering. Differences found are reported in the format described in the *pdfdiff*(1M) manual entry. (Size growth (**-p** option) is not reported.) For a detailed explanation of the PDF fields see *pdf*(4). The command

```
pdfck -r /pseudoroot /system/UX_CORE/pdf
```

is roughly equivalent to

```
mkpdf -r /pseudoroot /system/UX_CORE/pdf - | pdfdiff
/system/UX_CORE/pdf -
```

Options

pdfck recognizes the following options:

-n Compare numerical representation of user id *uid* and group id *gid* of each file, instead of the usual text representation. If owner or group is recorded in the PDF as a name, look the name up in the **/etc/passwd** or **/etc/group** file, respectively, to find the id number.

-r *alternate_root* *alternate_root* is a string that is prefixed to each pathname in the prototype when the filesystem is being searched for that file. Default is NULL.

EXAMPLES

The following output indicates tampering with **/bin/cat**:

```
/bin/cat: mode(-r-xr-xr-x -> -r-sr-xr-x)(became suid), size(27724 -> 10345),
        checksum(1665 -> 398)
```

FILES

/system/*fileset_name***/pdf**

SEE ALSO

mkpdf(1M), pdfdiff(1M), pdf(4).

NAME

pdfdiff - compare two Product Description Files

SYNOPSIS

pdfdiff [**-n**] [**-p** *percent*] *pdf1 pdf2*

DESCRIPTION

pdfdiff is a program that compares two PDFs (Product Description Files). The PDFs can be generated using the **mkpdf** command (see *mkpdf*(1M)). Individual fields in the PDFs are compared, and differences found in these fields are reported. For a detailed explanation of the PDF fields see *pdf*(4).

The report format is:

pathname: *diff_field*[(*details*)][,...]

diff_field is one of the field names specified in *pdf*(4). The format of *details* is "*oldvalue* -> *newvalue*" and may include an additional "(*added description*)".

A summary of total product growth in bytes, **DEV_BSIZE** disk blocks, and the percentage change in disk blocks is reported. This summary includes growth of all files, including those for which growth did *not* exceed the threshhold *percent*. Format of the growth summary is:

Growth: *x* bytes, *y* blocks (*z*%)

Options

pdfdiff recognizes the following options:

-n Compare numerical representation of user ID *uid* and group ID *gid* of each file, instead of the usual text representation. If owner or group is recorded in the PDF as a name, look the name up in the **/etc/passwd** or **/etc/group** file, respectively, to find the ID number.

-p *percent* specifies a threshhold percentage for file growth. Files having a net size change greater than or equal to this percentage are reported. A decrease in size is reported as a negative number. If **-p** is not specified, a default value of zero percent is used.

EXAMPLES

The following output results when the **/bin/cat** entry in the example from *pdf(4)* is different in the compared PDF:

```
/bin/cat: mode(-r-xr-xr-x -> -r-sr-xr-x)(became suid), size(27724 -> 10345),
      checksum(1665 -> 398)
Growth: -17379 bytes, -17 blocks (-4%)
```

FILES

/system/*fileset_name***/pdf**

SEE ALSO

mkpdf(1M), pdfck(1M), pdf(4).

NAME

perf - test the NCS RPC runtime library

SYNOPSIS

/etc/ncs/perf/server [**-d**] *max_calls family*

/etc/ncs/perf/run_client *family hostname*

/etc/ncs/perf/client *test_number test_args*

DESCRIPTION

The *perf* exerciser tests the functionality and measures the performance of the Remote Procedure Call (RPC) runtime library, part of the Network Computing System (NCS).

perf consists of client and server programs called **client** and **server** and a shell script called **run_client**. These executables reside in the directory **/etc/ncs/perf**.

The *perf* client and server programs run as separate processes, either on one host or on two different hosts. Any host that runs either the client or the server must have the Network Computing Kernel (NCK), the run-time portion of NCS, installed.

perf makes only minimal use of the Location Broker.

To use *perf*, first start the server, then run the client to conduct various NCK tests against that server, as described in the following sections.

Starting the perf Server

Any host that runs the *perf* server must also run **llbd**, the Local Location Broker daemon. Ensure that an **llbd** is running.

The *perf* server has the following syntax:

/etc/ncs/perf/server [**-d**] *max_calls family*

The **-d** option instructs the server to display debug messages.

The *max_calls* argument specifies the maximum number of calls that the server can execute in parallel. Unless your system provides Concurrent Programming Support (currently available only on Apollo Domain/OS systems), specify the value **1**.

The *family* argument specifies one or more address families for which the server should create sockets to listen on. On HP-UX systems, specify **ip**.

For testing NCK functionality, a typical invocation of the *perf* server is

```
$ server 1 ip
```

which creates a *perf* server that handles one call at a time, listens for *perf* client tests on an internet socket, and does not run in debug mode.

Running the run_client Script

The *perf* client consists of 11 tests, each of which exercises a particular feature of NCK.

To test NCK functionality, invoke the **run_client** script, which runs all 11 of the *perf* tests. (Several of the tests are run twice, once with the **idempotent** operation attribute and once without.)

The syntax for **run_client** is

$ **run_client** *family hostname*

The *family* and *hostname* arguments specify the host that is running the *perf* server.

Running Individual perf Client Tests

You can run the *perf* tests individually by directly invoking the **client** program.

Following are the individual *perf* client tests.

Test	Syntax
Null call	**client 0** *host passes calls / pass verify? idempotent?*
Variable length input argument	**client 1** *host passes calls / pass verify? idempotent? nbytes*

Variable length output argument	**client 2** *host passes calls/pass verify? idempotent? nbytes*
Broadcast	**client 3** *family*
Maybe	**client 4** *host*
Broadcast/maybe	**client 5** *family*
Floating point	**client 6** *host passes calls/pass verify? idempotent?*
Unregistered interface	**client 7** *host*
Forwarding	**client 8** *host global?*
Exception	**client 9** *host*
Slow call	**client 10** *host passes calls/pass verify? idempotent? seconds*
Shutdown	**client 11** *host*

Test 0 makes the simplest possible remote procedure call, one with no parameters.

Tests 1 and 2 pass open arrays as input and output parameters, respectively.

Tests 3, 4, and 5 use the **broadcast** and **maybe** operation attributes. (Tests 3 and 5 will not work if the server and the client are running on hosts in different networks.)

Test 6 passes floating-point parameters.

Test 7 requests an operation in an interface that is not registered with the RPC runtime library at the server.

Test 8 sends a call to the Location Broker forwarding port at the server, testing the forwarding facility provided by *llbd*. (Test 8 requires an *llbd* to be running at the server. The others specify well-known ports.)

Test 9 makes a call that raises an exception at the server.

Test 10 makes a call from which the server will return after the specified period of time.

Test 11 remotely shuts down the server.

Following are descriptions of the arguments that appear in the syntax table.

host The host on which the *perf* server is running, specified in the form *family:name* (for example, **ip:toscanini**).

passes
The number of times to run the test. The *perf* client displays a message after each pass.

calls/pass
The number of remote calls to make during one pass.

verify?
Yes/no (**y** or **n**) input. A "yes" input directs *perf* to run the test for correct operation. A "no" input directs *perf* to run the test to obtain performance statistics.

idempotent?
Yes/no (**y** or **n**) input. A "yes" input directs the *perf* client to call an idempotent operation at the *perf* server. A "no" input directs the *perf* client to call a non-idempotent operation.

nbytes
The number of bytes the input and output parameters should have.

family
An address family (for example, **ip** or **dds**).

global?
Yes/no (**y** or **n**) input. At this release, specify **n**.

seconds
The number of seconds the server should wait before returning.

Using perf to Troubleshoot NCK

If *perf* is not running successfully, check that the networking software and hardware used by NCK is functioning correctly. On networks that support IP protocols, for example, try running *telnet*, *ftp*, and other IP-based applications between the hosts that ran *perf*. Network services must be available and working

correctly before you start Location Brokers and other NCS-based programs.

If the *perf* tests appear to be running successfully but NCK appears to be functioning incorrectly, try reversing the roles of the *perf* hosts. It is possible that the *perf* tests will work correctly with the client running on host A and the server running on host B but will fail with the client running on host B and the server running on host A.

If any of the *perf* tests fail, try running the tests locally, with both the client and the server running on the same host. Failure in this case could be due to incorrect mapping of names to addresses (in a host table, for instance), absence of required daemons or servers, or mismatch of versions between the *perf* client and server.

SEE ALSO

lb_test(1M), llbd(1M).

NAME

ping - send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

ping [**-r**] [**-v**] [**-o**] *host* [*packetsize*] [*count*]

DESCRIPTION

ping sends an ICMP echo (ECHO_REQUEST) packet to *host* once per second. Each packet that is echoed back (via an ECHO_RESPONSE packet) is reported on the screen, including round-trip time.

ECHO_REQUEST datagrams ("pings") have an IP and ICMP header, followed by a **struct timeval**, and an arbitrary number of "pad" bytes used to fill out the packet. Default datagram length is 64 bytes, but this can be changed by using the command-line option.

Other options and parameters are:

-r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it (such as after the interface was dropped by **gated** (see *gated*(1M)).

-v Verbose output. ICMP packets other than ECHO_RESPONSE that are received are listed.

-o Insert "record route" IP option in outgoing packets, summarizing routes taken when the program exits. It may not be possible to get the round-trip path if all hosts on the route taken do not implement the "record route" IP option. A maximum of nine Internet addresses can be displayed due to the maximum length of the IP option area.

host *host* can be a hostname or an Internet address. All symbolic names specified for a host are looked up using **gethostbyname()** (see *gethostbyname*(3N)). If *host* is an Internet address, it must be in "dot" notation (see *inet_addr*(3N)).

packetsize By default (when *packetsize* is not specified), the size of transmitted packets is 64 bytes. The minimum value allowed for *packetsize* is eight bytes, and the maximum is 4096 bytes. Also, if *packetsize* is smaller than 16 bytes, there is not enough room for timing information. In this case the round-trip times are not displayed.

count The number of packets **ping** will transmit before terminating. **Range:** 1 to (2**31 -1), decimal. **Default:** **ping** sends packets until interrupted.

When using **ping** for fault isolation, it should first be run on the local host to verify that the local network interface is working correctly, then hosts and gateways further and further away should be **ping**ed. **ping** sends one datagram per second, and prints one line of output for every ECHO_RESPONSE returned. No output is produced if there is no response. If an optional *count* is given, only the specified number of requests is sent. Round-trip times and packet loss statistics are computed. When all responses have been received or the program times out (with a *count* specified), or if the program is terminated with a **SIGINT**, a brief summary is displayed.

This program is intended for use in testing, managing, and measuring network performance. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is considered discourteous to use **ping** unnecessarily during normal operations, or from automated scripts.

AUTHOR

ping was developed in the Public Domain.

FILES

/etc/hosts

SEE ALSO

gethostbyname(3N), rlb(1M), inet(3N).

NAME

portmap - DARPA port to RPC program number mapper

SYNOPSIS

`/etc/portmap`

DESCRIPTION

The `portmap` server converts RPC program numbers into DARPA protocol port numbers. It must be running to make RPC calls.

When an RPC server starts, it registers itself with `portmap`, telling `portmap` what port number it is listening to and what RPC program numbers it is prepared to serve. When a client wants to make an RPC call to a given program number, it first contacts `portmap` on the server machine to determine the port number where RPC packets should be sent.

The `portmap` server must be started before `inetd` because standard RPC servers are started by `inetd` (see *inetd*(1M)).

WARNINGS

If `portmap` crashes, all RPC servers must be restarted.

AUTHOR

`portmap` was developed by Sun Microsystems, Inc.

SEE ALSO

inetd(1M), rpcinfo(1M), inetd.conf(4).

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages

(Requires Optional LAN/X.25 Software)

NAME

proxy - manipulates the NS Probe proxy table

SYNOPSIS

proxy on

proxy off

proxy add *nodename domain ip_address medium*

proxy append *nodename domain ip_address medium*

proxy delete *nodename*

proxy show *nodename*

proxy flush

proxy list

DESCRIPTION

proxy enables or disables Probe proxy server capability on a node on a LAN or WAN. **proxy** also adds entries to the Probe proxy table or deletes entries from the Probe proxy table on a node. A probe proxy server enables NS through a LAN-to-LAN, LAN-to-WAN, WAN-to-WAN gateway when used with the *route*(1M) command. NS uses the probe protocol for name-to-IP-Address-resolution. By itself, the probe protocol can only obtain information about nodes on the same network or subnetwork. The probe proxy server provides IP address information about nodes on remote connected networks. The Probe proxy server can be a gateway node or any other node on the LAN. Note: The Probe protocol is not supported over X.25 links. In order to initiate NS over a connection path whose first hop is an X.25 link, the initiating node must be a Probe proxy server, and must include an entry for the target node in its Probe proxy table.

Options

on Enables Probe proxy on a node. This option must be used before invoking any other proxy option. Requires super-user capability.

off Disables the Probe proxy server. The Probe proxy table is not flushed. Requires super-user capability.

add Adds a new entry to the Probe proxy table. Requires super-user capability. The following parameters are required:

nodename A fully-qualified NS node name, including domain and organization. See *nodename*(1) for details.

domain Internet domain. The only supported domain is HPDSN.

ip_address An IP address of the remote node being mapped to by *nodename*. The IP address must be in internet "dot" format. See *inet*(3N) for details on internet dot format.

medium The physical link medium, either **ieee**, **ether**, or **x.25**.

append Appends an additional path report to an existing Probe proxy table entry. Use this option if a remote node supports more than one among IEEE 802.3 (**ieee**), Ethernet (**ether**), and X.25 (**x.25**) links, or if a remote node contains more than one network interface. **append** requires the same parameters as the **add** option described above. Requires super-user capability.

delete *nodename* Deletes *nodename* and all path reports associated with *nodename* from the Probe proxy table. Requires super-user capability.

flush Clears the Probe proxy table, deleting all entries. Requires super-user capability.

show *nodename* Sends path report information for *nodename* to standard output. For each path report associated with *nodename*, the following information is returned: node name, IP address or addresseses, medium, services, and transports. The services and transports field will always contain the value FFFF, meaning that all services and transports are enabled.

(Requires Optional LAN/X.25 Software)

list Returns information for all entries in the Probe proxy table. **proxy list** is equivalent to issuing a **proxy show** command for all node names in the Probe proxy table.

DIAGNOSTICS

proxy server not yet enabled -
must do so before issuing any other proxy commands
Other **proxy** options attempted before **proxy on** issued.

proxy on: proxy server was already enabled
proxy on issued after proxy server already enabled.

proxy off: proxy server was already disabled
Super-user tried to turn off an already disabled proxy server.

proxy add: entry already exists
Super-user tried to add to the proxy table an entry which already exists.

proxy: out of memory for path reports
Probe proxy memory account is depleted.

proxy add: no space left in hash table buckets
Super-user tried to add too many entries that hash to the same bucket in the proxy table.

proxy append: entry does not exist
Super-user tried to append to a proxy table entry which had not been added.

proxy delete: entry does not exist
Super-user tried to delete an entry in the proxy table which had never been added.

proxy show: no entry for *nodename*
Requested *nodename* had no entry in proxy table.

WARNINGS

Reciprocal *route* commands must be executed on the local proxy requestor, the destination host, and all intermediate hosts before NS routing can succeed. See *route*(1M) and *routing*(7) for details.

Proxy entries are hashed for storage. Each hash bucket contains room for only five entries. There are 19 hash buckets for a potential total of 95 entries.

AUTHOR

proxy was developed by HP

SEE ALSO

nodename(1), route(1M), inet(3N), routing(7).

NAME

pvchange - change characteristics of physical volume in a volume group

SYNOPSIS

/etc/pvchange -x *extensibility physical_volume_path*

DESCRIPTION

pvchange changes the characteristics and state of a physical volume in a volume group by setting the allocation permission for additional physical extents on the physical volume to either *allowed* or *prohibited*.

Command-Line Arguments

pvchange recognizes the following arguments:

-x *extensibility*

Set the allocation permission for additional physical extents on the physical volume specified by *physical_volume_path*. *extensibility* can have either of the following values:

y Allow allocation of additional physical extents on the physical volume.

n Prohibit allocation of additional physical extents on the physical volume. However, logical volumes residing on the physical volume are accessible.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **pvchange** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Prohibit allocation of additional physical extents to a physical volume:

```
pvchange -x n /dev/dsk/c2d0s2
```

Allow allocation of additional physical extents to a physical volume:

```
pvchange -x y /dev/dsk/c2d0s2
```

SEE ALSO

pvdisplay(1M).

NAME

pvcreate - create physical volume for use in a volume group

SYNOPSIS

/etc/pvcreate [**-b**][**-f**][**-B**][**-t** *disk_type*][**-d** *soft_defects*] *physical_vol_path*

DESCRIPTION

pvcreate initializes a direct access storage device (a raw disk device) for use as a physical volume in a volume group. *physical_vol_path* specifies the pathname of the raw device to be used.

If *physical_vol_path* contains a file system, **pvcreate** asks for confirmation if the **-f** option is not specified. Request for confirmation avoids accidentally deleting a file system. Currently, BSD and System V file systems are recognized.

The operation is denied if *physical_vol_path* belongs to another volume group. Only physical volumes not belonging to other volume groups can be created.

If *physical_vol_path* contains a disk label, it is updated to reflect that *physical_vol_path* is now a physical volume that can be installed in a volume group.

After using **pvcreate** to create a physical volume, use **vgcreate** to add it to a new volume group or **vgextend** or to add it to an existing volume group (see *vgcreate*(1M) and *vgextend*(1M)).

Raw devices cannot be added to a volume group until they are properly initialized by **pvcreate**.

physical_vol_path can be made a *bootable* disk by specifying the **-B** option, which reserves space on the physical volume for boot related data. This is a prerequisite for creating root volumes on logical volumes. Refer to *mkboot*(1M) and *lif*(4) for more information.

Options

pvcreate recognizes the following options:

-b Used to specify (on standard input) the numbers that correspond to the indexes of all known bad blocks on physical volume *physical_vol_path*, that is being created. Specify the indexes using decimal, octal, or hexadecimal numbers in standard C-language notation, with numbers separated by new-line, tab, or form-feed character. If this option is not used, **pvcreate** assumes that the physical volume contains no bad blocks.

-f Force creation of a physical volume (thus deleting any file system present) without first requesting confirmation. Currently BSD and System V file systems are recognized.

-t *disk_type* Retrieve configuration information about the physical volume from file **/etc/disktab**. *disk_type* specifies the device (hp7959S, for example).

This size only needs to be specified when **pvcreate** fails to get the size from the underlying disk driver. If the driver successfully returns the size of the device, *disk_type* is ignored.

-d *soft_defects* Used to specify minimum number of bad blocks that LVM should reserved in order to perform software bad block relocation. This number can be no larger than 7039. If not specified, one block is reserved for each 8K data blocks.

This option is not supported on HP-IB devices and *soft_defects* is set to 0 when **pvcreate** is executed for an HP-IB device.

-B Make a physical volume *bootable* (i.e. a **system** disk).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **pvcreate** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Create a physical volume on raw device **/dev/rdsk/c1d0s2**, and force the creation without confirmation:

```
pvcreate -f /dev/rdsk/c1d0s2
```

Create a physical volume on raw device **/dev/rdsk/c1d0s2**, specifying that a bad blocks list (7, 13, 95, and 133) must be read from standard input:

```
echo 7 13 95 133 | pvcreate -b /dev/rdsk/c1d0s2
```

FILES

/etc/disktab disk geometry and disk partition characteristics for all disk devices on the system

WARNINGS

Check the manufacturer's listing or run diagnostics testing for bad blocks on the device prior to creating a physical volume. If bad blocks are present, use the **-b** option when creating the physical volume.

SEE ALSO

mkboot(1M), vgcreate(1M), vgextend(1M), lif(4).

NAME

pvdisplay - display information about physical volumes within a volume group

SYNOPSIS

`/etc/pvdisplay` [`-v`] *physical_vol_path* ...

DESCRIPTION

`pvdisplay` displays information about the physical volume or volumes specified by the *physical_vol_path* parameter. If the `-v` (verbose) option is specified, `pvdisplay` displays a map of the logical extents that correspond to the physical extents of each physical volume.

-v not Specified on Command Line

If the `-v` option is absent, `pvdisplay` displays the characteristics of each physical volume specified by *physical_vol_path*, namely:

`PV Name`: Name of the physical volume

`VG Name`: Name of the volume group

`PV Status`: State of the physical volume: `available` or `unavailable`

`Allocatable`: Allocation permission for the physical volume

`VGDA`: Number of volume group descriptors on the physical volume

`Cur LV`: Number of logical volumes using the physical volume

`PE Size`: Size of physical extents on the volume

`Total PE`: Total number of physical extents on the physical volume

Free PE: Number of free physical extents on the physical volume

Allocated PE: Number of physical extents on the physical volume that are allocated to logical volumes

Stale PE: Number of physical extents on the physical volume that are not current

-v Specified on Command Line

If the `-v` is specified, `pvdisplay` lists additional information for each logical volume and for each physical extent on the physical volume:

`Distribution of physical volume`: Lists the logical volumes that have extents allocated on *physical_vol_path*:

LV Name: Name of the logical volume which has extents allocated on *physical_vol_path*.

LE of LV: Number of logical extents within the logical volume that are contained on this physical volume

PE for LV: Number of physical extents within the logical volume that are contained on this physical volume

`Physical extents`: Displays the following information for each physical extent:

`PE`: Physical extent number

`Status`: Current state of the physical extent: `free`, `used`, or `stale`

`LV`: Name of the logical volume to which the extent is allocated

`LE`: Index of the logical extent to which the physical extent is allocated

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **pvdisplay** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Display the status and characteristics of a physical volume:

```
pvdisplay /dev/dsk/c1d0s2
```

Display the status, characteristics, and allocation map of a physical volume:

```
pvdisplay -v /dev/dsk/c2d0s2
```

SEE ALSO

lvdisplay(1M), pvchange(1M), vgdisplay(1M).

NAME

pvmove - move allocated physical extents from one physical volume to one or more other physical volumes

SYNOPSIS

/etc/pvmove [**-n** *lv_path*] *source_physical_vol_path* [*dest_physical_vol_path* ... | *dest-physical-vol-group-name*...]

DESCRIPTION

pvmove moves allocated physical extents and the data they contain from source physical volume *source_physical_vol_path* to one or more other physical volumes. To limit the transfer to specific physical volumes, specify the names of one or more physical volumes with the *dest_physical_vol_path* or indirectly via the *dest-physical-vol-group-name* parameter if physical volume groups have been defined under the volume group that contains the source physical volume.

If the destination volume or volumes are not specified, all physical volumes in the volume group are available as destination volumes for the transfer. However, LVM selects the proper physical volumes to be used in order to preserve the allocation policies of the logical volume involved. All destination physical volumes must be within the same volume group. The *source_physical_vol_path* must not appear in the *dest_physical_vol_path* parameter. If the source physical volume is a member of the destination physical volume group, it is automatically excluded from being a destination physical volume.

pvmove succeeds only if there is enough space on the *dest_physical_vol_path* or on the *dest-physical-vol-group-name* to hold all the extents of the *source_physical_vol_path*.

Options

pvmove recognizes the following options and accompanying parameters:

-n *lv_path*
Moves only physical extents allocated to the logical volume (specified by *lv_path*) that are located on the source physical volume (specified by *source_physical_vol_path*) to the specified destination physical volume.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **pvmove** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Move physical extents from **/dev/dsk/c1d0s2** to **/dev/dsk/c2d0s2** and **/dev/dsk/c3d0s2**:

```
pvmove /dev/dsk/c1d0s2 /dev/dsk/c2d0s2 /dev/dsk/c3d0s2
```

If **/dev/dsk/c2d0s2** and **/dev/dsk/c3d0s2** belong to **PVG0**, the same result can be achieved using the following command:

```
pvmove /dev/dsk/c1d0s2 PVG0
```

Move only the physical extents in logical volume **/dev/vg01/lvol2** that are on **/dev/dsk/c1d0s2** to **/dev/dsk/c2d0s2**:

```
pvmove -n /dev/vg01/lvol2 /dev/dsk/c1d0s2 /dev/dsk/c2d0s2
```

SEE ALSO

pvdisplay(1M).

NAME

pwck, grpck - password/group file checkers

SYNOPSIS

/etc/pwck [*file*]
/etc/grpck [*file*]

DESCRIPTION

pwck scans the default password file or *file* and reports any inconsistencies to standard error. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The criteria for determining a valid login name are described in the *HP-UX System Administrator* manuals for your system. The default password file is **/etc/passwd**.

grpck verifies all entries in the group file and reports any inconsistencies to standard error. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group**.

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

AUTHOR

pwck was developed by AT&T and HP.

DEPENDENCIES

NFS:

pwck and **grpck** check only the local password and group files. The Network Information Service database for password and group files is not checked.

FILES

/etc/group
/etc/passwd

SEE ALSO

group(4), passwd(4).

STANDARDS CONFORMANCE

pwck: SVID2

grpck: SVID2

NAME

quot - summarize file system ownership

SYNOPSIS

`/etc/quot` [`-cfhnv`] *filesystem*

`/etc/quot -a` [`-cfhnv`]

DESCRIPTION

`quot` displays the number of 1024-byte blocks in the named *filesystem* that are currently owned by each user. *filesystem* is either the name of the directory on which the file system is mounted or the name of the device containing the file system.

Options

quot recognizes the following options:

`-a` Generate a report for all mounted file systems.

`-c` Report size rather than user statistics. Generates histogram statistics in 3-column format:

Column 1: File size in blocks. Sizes are listed in ascending order up to 499 blocks per file. Files occupying 499 or more blocks are counted together on a single line as 499-block files (but column 3 is based on actual number of blocks occupied).

Column 2: Number of files of size indicated in column 1.

Column 3: Cumulative total blocks occupied by files counted in current plus all preceding lines.

Use of this option overrides the `-f` and `-v` options.

`-f`
Display number of files and space occupied by each user.

`-h`
Calculate the number of blocks in the file based on file size rather than actual blocks allocated. This option does not account for sparse files (files with holes in them).

`-n`
Accept *ncheck*(1M) data as input. Run the pipeline:

`ncheck` *device* `| sort +0n | quot -n` *filesystem*

to produce a list of all files and their owners.

`-v`
Display three columns containing the number of blocks not accessed in the last 30, 60, and 90 days.

AUTHOR

Disk Quotas were developed by the University of California, Berkeley, Sun Microsystems, Inc., and HP.

FILES

`/etc/mnttab`	mounted file systems
`/etc/passwd`	user names

SEE ALSO

du(1), find(1), ls(1), mount(1M), repquota(1M), quota(5).

NAME

quotacheck - file system quota consistency checker

SYNOPSIS

/etc/quotacheck [**-v**] [**-p**] [**-P**] filesystem ...
/etc/quotacheck [**-v**] [**-p**] [**-P**] **-a**

DESCRIPTION

quotacheck examines each file system, builds a table of current disk usage, and compares this table against that stored in the disk quota file for the file system. If any inconsistencies are detected, both the quota file and the current system copy of the incorrect quotas are updated.

quotacheck expects each file system to be checked to have a file named **quotas** in the root directory. If none is present, **quotacheck** reports an error and ignores the file system. **quotacheck** is normally run at mount time from file **/etc/rc**. See WARNINGS below.

Options

quotacheck recognizes the following options:

-a Obtain list of file systems to check from **/etc/checklist**. Only mounted **rw** type file systems with the **quota** option are checked.

-v Indicate the calculated disk quotas for each user on a particular file system.

-p Check file systems in parallel as allowed by equal values in the *pass number* field in **/etc/checklist**.

-P Preen file systems, checking only those with invalid quota statistics (**quotaoff** and **edquota** commands can invalidate quota statistics as discussed in *quota*(5) — see (*quotaoff*(1M) and *edquota*(1M)). Also checks in parallel as in **-p** above.

AUTHOR

Disk Quotas were developed by the University of California, Berkeley, Sun Microsystems, Inc., and HP.

FILES

/etc/checklist default file systems
/etc/mnttab information about mounted file systems
*directory***/quotas** quota statistics static storage for filesystem where *directory* is the file system root as specified to **mount** (see *mount*(1M)).

SEE ALSO

rc(1M), quota(5).

NAME

quotaon, quotaoff - turn file system quotas on and off

SYNOPSIS

/etc/quotaon [**-v**] *file_system* ...
/etc/quotaon [**-v**] **-a**
/etc/quotaoff [**-v**] *file_system* ...
/etc/quotaoff [**-v**] **-a**

Remarks

These commands are provided for compatibility only. Their use is generally neither required nor recommended because **mount** and **umount** enable and disable quotas cleanly (see *mount*(1M) and *umount*(1M)). See WARNINGS below for more information.

DESCRIPTION

quotaon announces to the system that disc quotas should be enabled on one or more file systems. *file_system* is the name of either the directory on which the file system is mounted or the block special device. The file systems specified must have entries in **/etc/checklist** and be mounted at the time. The file system quota file, named **quotas**, must be present in the root directory of each specified file system.

When enabling quotas interactively after boot time, the **quotacheck** command should be run first (see Remarks above and WARNINGS below).

quotaoff announces to the system that file systems specified should have any disc quotas turned off.

Use **mount** to determine the current state of quotas on mounted file systems.

Options

The following options are recognized:

-v Obtain the *file_system* list from **/etc/checklist** of type **rw** with the **quota** option (see *checklist*(4)).

-a Generate a message for each file system affected.

WARNINGS

Using **quotaoff** to disable quotas on a file system causes the system to discontinue tracking quotas for that file system, and marks the **quota clean** flag in the super block **NOT_OK**. This, in turn, forces a **quotacheck** the next time the system is booted. Since quotas are enabled and disabled cleanly by **mount** and **umount** anyway, use of **quotaon** and **quotaoff** is generally discouraged.

AUTHOR

Disk quotas were developed by the University of California, Berkeley, Sun Microsystems, Inc., and HP.

FILES

/etc/checklist	file system table
/etc/mnttab	mount table
directory/**quotas**	quota statistics storage for file system where *directory* is the root of the file system as specified to **mount**.

SEE ALSO

mount(1M), quota(5).

NAME

rbootd - remote boot server

SYNOPSIS

/etc/rbootd [**-a**] [**-l** *loglevel*] [**-L** *logfile*] [*logfile*]

DESCRIPTION

rbootd services initial boot-up requests from HP Cluster clients and Datacommunications and Terminal Controllers (DTC/9000) over a local area network. *rbootd* only responds to requests from machines that are listed in the cluster configuration file or map802 file. The map802 file (a binary file) is created when a DTC is configured by *dtcconfig* on the host machine.

For HP Cluster clients, *rbootd* emulates part of the context of a client machine based on information provided by the client in the request and on information found in the cluster configuration file. Therefore, bootstrap programs and kernel programs requested by the client machine can be context-dependent files.

The following elements appear in the context emulated by *rbootd* for HP cluster clients:

cnode name
architecture
remoteroot
default

where *cnode name* is based on information found in the cluster configuration file.

The *architecture* field is based on the identity provided to *rbootd* by the booting client, according to the following table:

Machine Type	*architecture* Field
Series 300, 400	HP-MC68881 HP-MC68020 HP-MC68010
Series 700	PA-RISC1.1 HP-PA
Series 800	HP-PA

Options

rbootd supports the following options:

-a Append to the *rbootd* log file. By default, starting up *rbootd* truncates the log file.

-l *loglevel* Set the amount of information that will be logged in the log file. *rbootd* supports the following logging levels:

0 Log only startup and termination messages of *rbootd*.
1 Log all errors. This is the default logging level.
2 Log rejected boot requests from machines not found in **/etc/clusterconf** or **/usr/dtcmgr/map802**.
3 Log all boot requests.

-L *logfile*
Specify an alternate file that *rbootd* should use to log status and error messages.

landevice Specify an alternate device that *rbootd* should use to listen for boot requests.

AUTHOR

rbootd was developed by HP.

FILES

/dev/ieee	default local area network device
/etc/clusterconf	cluster configuration file
/usr/adm/rbootd.log	default *rbootd* log file
/usr/boot/*	bootstrap programs
/etc/boottab	bootstrap configuration file
/usr/dtcmgr/map802	DTC/9000 configuration file

SEE ALSO

cdf(4), clusterconf(4), context(5), dtcconfig(1M), dtcnmd(1M), dtcnmp(1M).

NAME

rcancel - remove requests from a remote line printer spooling queue

SYNOPSIS

/usr/lib/rcancel [*id* ...] [*printer*] [**-a**] [**-e**] [**-u** *user*]

DESCRIPTION

rcancel removes a request, or requests, from the spool queue of a remote printer. **rcancel** is invoked by the **cancel** command (see *cancel*(1)).

At least one *id* or the name of a *printer* must be specified.

This command is intended to be used only by the spool system in response to the **cancel** command (see *lp*(1)), and should not be invoked directly.

Options

rcancel recognizes the following options:

id Specifying a request ID (as returned by **lp** (see *lp*(1)) cancels the associated request (if the request is owned by the user), even if it is currently printing.

printer Name of the printer (for a complete list, use **lpstat** — see *lpstat*(1)). Specifying a *printer* cancels the request which is currently printing on that printer, if the request is owned by the user. If the **-a**, **-e**, or the **-u** option is specified, this option only specifies the printer to perform the cancel operation on.

-a Remove all requests owned by the user on the specified printer (see *printer*). The owner is determined by the user's login name and host name on the machine where the **lp** command was invoked.

-e Empty the spool queue of all requests for the specified *printer*. This form of invoking **rcancel** is useful only to users with appropriate privileges.

-u *user* Remove any requests queued belonging to that user (or users). This form of invoking **rcancel** is available only to users with appropriate privileges.

HP Clustered Environment

In the HP Clustered Environment, all spooling is handled as if the cluster nodes were a single system and all printers attached to either the cluster server or clients can be available. Remote spooling applies to spooling from or to machines outside of the cluster nodes.

AUTHOR

rcancel was developed by the University of California, Berkeley, and HP.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1), rlp(1M), rlpdaemon(1M), rlpstat(1M).

NAME

reboot - reboot the system

SYNOPSIS

/etc/reboot [**-h**|**-r**] [**-n**|**-s**] [**-q**] [**-t** *time*] [**-m** *message*] [**-d** *device*] [**-f** *lif_filename*] [[**-l** *server_linkaddress*] | [**-b** *boot_server*]]

DESCRIPTION

reboot terminates all currently executing processes except those essential to the system, then halts or reboots the system. When invoked without arguments, **reboot** syncs all disks before rebooting the system.

Options

reboot recognizes the following options:

-h Shut down the system and halt.

-r Shut down the system and reboot automatically (default).

-n Do not sync the file systems before shutdown.

-s Sync the file systems before shutdown; for file systems that were cleanly mounted, modify the **fs_clean** flag from **FS_OK** to **FS_CLEAN** (default).

-q Quick and quiet. Suppress broadcast of warning messages, terminate processes by brute force (with **SIGKILL**) and immediately call **reboot()** with arguments as indicated by the other options (see *reboot*(2)). No logging is performed. The **-t** and **-m** options are ignored with this option.

-t *time* Specify what time **reboot** will bring the system down. *time* can be the word **now** (indicating immediate shutdown) or a future time in one of two formats: *+number* and *hour*:*min*. The first form brings the system down in *number* minutes; the second brings the system down at the time of day indicated (based on a 24-hour clock).

-m *message* Display *message* at the terminals of all users on the system at decreasing intervals as reboot *time* approaches. The *message* must not contain any embedded double quotes.

-d *device* (Series 300 and 400 only) Reboot from the specified device. The device must be a LIF volume or LAN interface. This option cannot be used with **-h**.

-f *lif_filename* (Series 300 and 400 only) Reboot from the specified file. If the filename is an empty string, the power-up search sequence is made for a system. Otherwise, the file name has to follow the LIF filename convention (see *lif*(4)). This option cannot be used with **-h**.

-l *server_linkaddress*
(Series 300 and 400 only) Use the system identified by this address as the new boot server. This is the ETHERNET link address of the LAN interface card; for example, **0x080009006997**. This number must be in a format acceptable to **reboot()**. If the new server is on a LAN connected to a different interface, the **-d** option also must be specified. The named system must be an active HP Cluster server, properly configured to serve this cluster node.

-b *boot_server* (Series 300 and 400 only) Use system *boot_server* as the new boot server. The named system must be an active HP Cluster server, properly configured to serve this cluster node, and listed in the file **/etc/bootservers**. Entries in this file have the following syntax:

server_name server_linkaddress [*device*] # *comment*

The **-l** and **-b** options cannot be used together.

At shutdown time a message is written in the file **/usr/adm/shutdownlog** (if it exists), containing the time of shutdown, who ran **reboot**, and the reason.

In the HP Clustered environment, executing **reboot** on the cluster server causes all of the cluster nodes to be rebooted. Executing it on a swap server causes all swap clients to be rebooted. Executing **reboot** on a cluster client reboots the client system. Arguments for local **reboot** commands are copied from the

server's **reboot** command, with the exception of the **-n**, **-q**, **-d**, and **-f** options. Using the **-l** or **-b** option on the cluster server causes all members of the cluster, including the cluster server, to reboot using the new server. On a swap server client system, all swap clients and the swap server are rebooted using the new server. On a client system that is not a swap server, only the client is rebooted, and the effect is to change cluster servers.

Executing **reboot** on a client node that is a swap server affects that node and all of its swap clients. Executing **reboot** on a client node that is not a swap server only affects that node.

Only users with appropriate privileges can execute the **shutdown** command.

DEPENDENCIES

Series 700/800

The **-b**, **-d**, **-f**, and **-l** options are not supported.

AUTHOR

reboot was developed by HP and the University of California, Berkeley.

FILES

/usr/adm/shutdownlog	shutdown log
/etc/bootservers	table of system names usable with the **-b** option.

SEE ALSO

lif*(1), reboot(2), lif(4), privilege(5).

NAME

recoversl - check and recover damaged or missing shared libraries

SYNOPSIS

`/etc/recoversl [-f]`

DESCRIPTION

`recoversl` is a command run by `init` at boot time to check for the existence of shared libraries that are critical to the system (see *init*(1M)). `recoversl` checks for, and corrects if necessary, the owner, group and permissions of these shared libraries.

Critical shared libraries include: `dld.sl` (the dynamic loader), and `libc.sl`. If any of these libraries are missing or damaged, `recoversl` guides the system administrator through a set of procedures that are designed to recover the shared libraries from update media.

`recoversl` is designed to be run at boot time, but can be run by a user with appropriate privileges at any time when critical shared library problems are suspected.

Options

`-f` Force recovery of critical shared libraries, regardless of status of sanity checks.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

AUTHOR

`recoversl` was developed by HP.

SEE ALSO

tar(1), mknod(1M), mount(1M), dld.sl(5).

NAME

regen - regenerate (uxgen) an updated HP-UX system

SYNOPSIS

regen [**-A**] [**-B**] [**-C**] [**-D**] [**-E**] [**-F**] [**-G**] [**-I**] [**-K**] [**-M**] [**-O**] [**-P**] [**-S**] [**-W**] [**-X**] [**-ZBSDIPC-SOCKET**] [**-ZNETIPC**] [**-ZNETINET**] [**-ZNET**] [**-ZLAN**] [**-ZAPPLTALK**] [**-c**] [**-i**] [**-l**] [**-p**]

DESCRIPTION

regen is a utility used during the *update*(1M) process to aid the user in generating a new Series 800 HP-UX kernel.

Only the super-user can execute *regen*.

By default *regen* is interactive. If interactive, the user is asked if the configuration file **/etc/conf/gen/S800** matches the system's I/O configuration and options. If it does not, *regen* proceeds to create a configuration file which does match the system.

Unless the **-p** option is used, *regen* changes the configuration file according to the options specified on the command line and in the file **/etc/conf/gen/.regenrc**. When the customize scripts are run during an update, each product that affects the kernel drops a flag into the **.regenrc** file. This flag tells *regen* to include that product's functionality in the configuration file. If a product's flag is not included in **.regenrc** or on the command line, its functionality is commented out of the configuration file, if present In that case *regen* assumes the product's libraries have not been updated and whatever is in **/etc/conf/lib** may be incompatable with the core libraries. If the running kernel is not **/hp-ux**, **/hp-ux** does not exist, or if **/hp-ux** is unreadable, *regen* may have to ask the user for information about the system. For example, *regen* may ask what type of root disk or console your system has.

After *regen* has modified the configuration file, it calls *uxgen*(1M) to build a kernel using **/etc/conf/gen/S800** as input. If the *uxgen* succeeds, *regen* saves the old kernel in **/SYSBCKUP**, and moves the new one from **/etc/conf/S800/hp-ux** to **/hp-ux**. If the **-i** option is used, *regen* finally does an *insf*(1M) and copies an **/etc/newconfig/inittab** into **/etc/inittab.**

Options

-A Absolutely no interaction is allowed. Regen exits with a status of 1 if it cannot create a kernel without asking the user questions.

-B The BX25 product has been loaded. Include the BX25 subsystem.

-C The X25 product has been loaded. Include the X25 subsystem.

-D Reserved.

-E The X25-IP product is to be activated. Include the X25 subsystem with the X25-IP fileset (this flag must be specified with **-C**).

-F The NFS product has been loaded. Include the NFS subsystem.

-G The G2 product has been loaded. Include the G2 subsystem.

-I Non-interactive mode. The file **/etc/conf/gen/S800** is modified according to the options specified at the command line and in the **.regenrc** file. No questions are asked.

-K The Datakit product has been loaded. Include the Datakit subsystem.

-M The Mirror Disk product has been loaded. Include the Mirrored Disk subsystem.

-O The OSI Link product has been loaded. Include the OSI Link subsystem.

-P The SWITCHOVER product has been loaded.

-S The Fairshare Scheduler product has been loaded. Include the Fairshare Scheduler subsystem.

-W Reserved.

-X The XT product has been loaded. Include the XT subsystem.

-ZBSDIPC-SOCKET
The UNIX Domain sockets fileset has been loaded. Include the BSDIPC-SOCKET fileset (*).

-ZNETIPC
The NetIPC fileset has been loaded. Include the NETIPC fileset (*).

-ZNETINET
The NETINET fileset has been loaded. Include the NETINET fileset (*).

-ZNET
The NET fileset has been loaded. Include the NET fileset (*).

-ZLAN
The LAN fileset has been loaded. Include the LAN fileset (*).

-ZAPPLTALK
The APPLTALK fileset has been loaded. Include the APPLTALK fileset (*).

-c If the user claims the file **/etc/conf/gen/S800** matches the I/O configuration of his system, check that the *console*, *root*, *swap*, *dumps*, and *args* devices match the running kernel's. If they do not, *regen* exits with a status of 1.

-i Install mode. An S800 file is always created.

-l Reserved.

-p Plain mode. If the user claims the file **/etc/conf/gen/S800** matches the system I/O configuration, the file is used exactly as is. *regen* does not modify it according to the options specified on the command line or in the **.regenrc** file.

(*) Supported only in specific combinations. Please see "Installing and Administering *LAN/9000* and other networking manuals to determine supported combinations.

DIAGNOSTICS
Exit status is 0 if *regen* executes successfully and the new kernel is in place. Exit status is 2 if the user quits *regen* prematurely. If any part of *regen* fails, exit status is 1.

DEPENDENCIES
regen is not implemented on Series 700.

AUTHOR
regen was developed by HP.

FILES
/etc/conf/gen/.regenrc
/etc/master
/etc/newconfig/inittab.ite
/etc/newconfig/inittab.mux
/etc/inittab

SEE ALSO
uxgen(1M), insf(1M), lssf(1M), mksf(1M), shutdown(1M).

NAME

remshd - remote shell server

SYNOPSIS

`/etc/remshd [-ln]`

DESCRIPTION

remshd is the server for the **rcp** and **remsh** commands and the **rcmd()** function (see *rcp*(1), *remsh*(1), and *rcmd*(3N)). The server provides remote execution facilities with authentication based on privileged port numbers.

inetd calls **remshd** when a service request is received at the port indicated for the **shell** (or **cmd**) service specified in **/etc/services** (see *inetd*(1M) and *services*(4)). **inetd** creates a connection to the service on the client's host. To run **remshd**, the following line should be present in **/etc/inetd.conf**:

```
shell   stream   tcp   nowait   root /etc/remshd   remshd
```

When **remshd** receives a service request, it responds with the following protocol:

1. The server checks the client's source port. If the port is not in the range 512 through 1023, the server aborts the connection.
2. The server reads characters from the connection up to a null (**\0**) byte. It interprets the resulting string as an ASCII number, base 10.
3. If the number is non-zero, it is interpreted as the port number of a secondary stream to be used for standard error. A second connection is then created to the specified port on the client's host. The source port of this second connection must be in the range 0 through 1023. If the first character sent is a null (**\0**), no secondary connection is made, and command standard error is sent to the primary stream. If the secondary connection has been made, **remshd** interprets bytes it receives on that socket as signal numbers and passes them to the command as signals. See *signal*(2).
4. The server checks the client's source address and requests the corresponding host name (see *gethostbyaddr*(3N), *hosts*(4), and *named*(1M)). If it cannot determine the hostname, it uses the dot-notation representation of the host address.
5. The server reads the client's host account name from the first connection. This is a null-terminated sequence not exceeding 16 characters.
6. The server reads the server's host account name from the first connection. This is a null-terminated sequence not exceeding 16 characters.
7. The server reads a command to be passed to the shell from the first connection. The command length is limited by the maximum size of the system's argument list.
8. **remshd** then validates the user as follows:

   The user account name for the server's host (step 6) is looked up in the password file and a **chdir()** is performed to the user's home directory in the server's host. If either the lookup or **chdir()** fails, the connection is terminated (see *chdir*(2)). If the client account is not equivalent to the server's host account, the connection is terminated. For more information on equivalent accounts see *hosts.equiv*(4).

9. A null byte is returned on the connection associated with standard error and the command line is passed to the normal login shell of the user with that shell's **-c** option. The shell inherits the network connections established by **remshd** and assumes the normal user and group permissions of the user.

   remshd uses the following path when executing the specified command:

   ```
   :/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin
   ```

10. If a secondary socket has been set up, **remshd** normally exits when command standard error and secondary socket standard error have both been closed. If no secondary socket was set up, **remshd** has **exec**ed the command and is no longer present (see *exec*(2)).

The `-l` option prevents any authentication based on the user's `.rhosts` file unless the user is the super-user.

Transport-level keep-alive messages are enabled unless the `-n` option is present. The use of keep-alive messages allows sessions to be timed out if the client crashes or becomes unreachable.

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with standard error after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps before the command execution).

`Malformed from address`
The first socket connection does not use a reserved port or the client's host address is not an ARPA Internet address.

`Can't get stderr port`
Unable to complete the connection of the secondary socket used for error communication.

`Second port not reserved`
The secondary socket connection does not use a reserved port.

`Locuser too long`
The name of the user account on the client's host is longer than 16 characters.

`Remuser too long`
The name of the user on the server's host is longer than 16 characters.

`Command too long`
The command line passed exceeds the size of the argument list (as configured into the system).

`Login incorrect`
No password file entry existed for the user name on the server's host, or the authentication procedure described above in step 8 failed.

`No remote directory`
The `chdir` command to the home directory in the server's host failed.

`Can't make pipe`
The pipe needed for the standard error output wasn't created.

`No more processes`
The server was unable to fork a process to handle the incoming connection.

Next step: Wait a period of time and try again. If this message persists, the server's host may have runaway processes that are using all the entries in the process table.

system call`: ...`
Error in executing the named system call. Appended to this error is a message specifying the cause of the failure.

shellname`: ...`
The user's login shell could not be started. This message is returned on the connection associated with the standard error, and is not preceded by a leading byte with a value of 1. Other messages can be returned by the remote command when it executes.

WARNINGS

The "privileged port" authentication procedure used here assumes the integrity of each host and the connecting medium. This is insecure, but is useful in an "open" environment.

`remshd` ignores `SIGHUP`, `SIGINT`, `SIGQUIT`, and `SIGTERM`, so these signal numbers can safely be sent to remote commands via `remshd`'s secondary socket. Other signal numbers may cause `remshd` to kill itself.

AUTHOR

`remshd` was developed by the University of California, Berkeley.

FILES

`/etc/hosts.equiv` list of equivalent hosts

`$HOME/.rhosts` user's private equivalence list

SEE ALSO

remsh(1), inetd(1M), named(1M), rcmd(3N), hosts(4), hosts.equiv(4), inetd.conf(4), inetd.sec(4), services(4).

NAME

repquota - summarize quotas for a file system

SYNOPSIS

repquota [**-v**] **-a**

repquota [**-v**] *filesystem* ...

DESCRIPTION

repquota prints a summary of disk usage and quotas for the specified file systems. For each user, the current number of files and amount of space (in Kbytes) is printed, along with any quotas created with **edquota** (see *edquota*(1M)).

Options:

repquota recognizes the following command-line options:

-a Report on all appropriate file systems in */etc/checklist*.

-v Report all quotas, even if there is no usage.

AUTHOR

Disk Quotas were developed by the University of California, Berkeley, Sun Microsystems, and HP.

FILES

/etc/checklist	default file systems
/etc/mnttab	information on mounted file systems
directory/**quotas**	quota statistics static storage for file system where *directory* is the root of the file system as interpreted by **mount** (see *mount*(1M)).

SEE ALSO

mount(1M), quota(5).

NAME

restore, rrestore - restore file system incrementally, local or across network

SYNOPSIS

/etc/restore *key* [*name* ...]
/etc/rrestore *key* [*name* ...]

DESCRIPTION

restore and *rrestore* read tapes dumped with the *dump*(1M) or *rdump*(1M) command. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying the files that are to be restored. Unless the **h** modifier is specified (see below), the appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The tape is read and loaded into the current directory. This should not be done lightly; **r** should be used only to restore a complete dump tape onto a clear file system or to restore an incremental dump tape after a full-level zero restore. Thus,

```
/etc/newfs /dev/rdsk/c0d0s10 hp7933
/etc/mount /dev/dsk/c0d0s10 /mnt
cd /mnt
restore r
```

is a typical sequence to restore a complete dump. Another *restore* or *rrestore* can be done to get an incremental dump in on top of this. Note that *restore* and *rrestore* leave a file *restoresymtab* in the root directory of the filesystem to pass information between incremental restore passes. This file should be removed when the last incremental tape has been restored. A *dump*(1M) or *rdump*(1M) followed by a *newfs*(1M) and a *restore* or *rrestore* are used to change the size of a file system.

R

restore and *rrestore* request a particular tape of a multi-volume set on which to restart a full restore (see **r** above). This allows *restore* and *rrestore* to be interrupted and then restarted.

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, and the **h** modifier is not specified, the directory is recursively extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the root directory is extracted, which results in the entire contents of the tape being extracted, unless **h** has been specified.

t Names of the specified files are listed if they occur on the tape. If no file argument is given, the root directory is listed, which results in the entire content of the tape being listed, unless **h** has been specified.

s

The next argument to *restore* is used as the dump file number to recover. This is useful if there is more than one dump file on a tape.

i This mode allows interactive restoration of files from a dump tape. After reading in the directory information from the tape, *restore* and *rrestore* provide a shell-like interface that allows the user to move around the directory tree selecting files to be extracted. The available commands are given below; for those commands that require an argument, the default is the current directory.

add [*arg*] The current directory or specified argument is added to the list of files to be extracted. If a directory is specified, it and all its descendents are added to the extraction list (unless the **h** key is specified on the command line). File names on the extraction list are displayed with a leading * when listed by **ls**.

cd [*arg*] Change the current working directory to the specified argument.

delete [*arg*] The current directory or specified argument is deleted from the list of files to be extracted. If a directory is specified, it and all its descendents are deleted from the extraction list (unless **h** is specified on the command line). The most expedient way to extract most files from a directory is to add the directory to the extraction list, then delete unnecessary files.

extract All files named on the extraction list are extracted from the dump tape. *restore* and *rrestore* will ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work toward the first volume.

help List a summary of the available commands.

ls [*arg*] List the current or specified directory. Entries that are directories are displayed with a trailing /. Entries marked for extraction are displayed with a leading *. If the verbose key is set, the inode number of each entry is also listed.

pwd Print the full pathname of the current working directory.

quit *restore* and *rrestore* immediately exit, even if the extraction list is not empty.

set-modes Set the owner, modes, and times of all directories that are added to the extraction list. Nothing is extracted from the tape. This setting is useful for cleaning up after a restore aborts prematurely.

verbose The sense of the **v** modifier is toggled. When set, the verbose key causes the **ls** command to list the inode numbers of all entries. It also causes *restore* and *rrestore* to print out information about each file as it is extracted.

The following function modifier characters can be used in addition to the letter that selects the function desired:

b Specify the block size of the tape in Kbytes. If the **-b** option is not specified, *restore* and *rrestore* try to determine the tape block size dynamically.

f Specify the name of the archive instead of **/dev/rmt/0m**. If the name of the file is **-**, *restore* reads from standard input. Thus, *dump*(1M) and *restore* can be used in a pipeline to dump and restore a file system with the command

dump 0f - /usr | (cd /mnt; restore xf -)

When using *rrestore*, this key should be specified, and the next argument supplied should be of the form *machine:device*.

h Extract the actual directory, rather than the files to which it refers. This prevents hierarchical restoration of complete subtrees from the tape, rather than the files to which it refers.

m Extract by inode numbers rather than by file name. This is useful if only a few files are being extracted and one wants to avoid regenerating the complete pathname to the file.

v Type the name of each file *restore* and *rrestore* treat, preceded by its file type. Normally *restore* and *rrestore* do their work silently; the **v** modifier specifies verbose output.

y Do not ask whether to abort the operation if *restore* and *rrestore* encounters a tape error. *restore* and *rrestore* attempt to skip over the bad tape block(s) and continue as best they can. *restore* and *rrestore* do not ask whether to abort the restore if they get a tape error. They attempt to skip over the bad tape block(s) and continue as best they can.

rrestore creates a server, */etc/rmt*, on the remote machine to access the tape device.

DIAGNOSTICS

restore and *rrestore* complain about bad key characters.

restore and *rrestore* complain if a read error is encountered. If the **y** modifier has been specified, or the user responds **y**, *restore* and *rrestore* attempt to continue the restore.

If the dump extends over more than one tape, *restore* and *rrestore* ask the user to change tapes. If the **x** or **i** function has been specified, *restore* and *rrestore* also ask which volume the user wants to mount. The fastest way to extract a few files is to start with the last volume and work towards the first volume.

There are numerous consistency checks that can be listed by *restore* and *rrestore*. Most checks are self-explanatory or can "never happen". Here are some common errors:

filename: **not found on tape**
The specified file name was listed in the tape directory but not found on the tape. This is caused

by tape read errors while looking for the file, and from using a dump tape created on an active file system.

expected next file *inumber*, **got** *inumber*
A file not listed in the directory showed up. This can occur when using a dump tape created on an active file system.

Incremental tape too low
When doing an incremental restore, a tape that was written before the previous incremental tape, or that has too low an incremental level has been loaded.

Incremental tape too high
When doing an incremental restore, a tape that does not begin its coverage where the previous incremental tape left off, or that has too high an incremental level has been loaded.

Tape read error while restoring *filename*
Tape read error while skipping over inode *inumber*
Tape read error while trying to resynchronize
A tape-read error has occurred. If a file name is specified, the contents of the restored files are probably partially wrong. If restore is skipping an inode or is trying to resynchronize the tape, no extracted files are corrupted, although files may not be found on the tape.

Resync restore, skipped *num* **blocks**
After a tape-read error, *restore* and *rrestore* may have to resynchronize themselves. This message indicates the number of blocks skipped over.

WARNINGS

restore and *rrestore* can get confused when doing incremental restores from dump tapes that were made on active file systems.

A level-zero dump (see *dump*(1M)) must be done after a full restore. Since restore runs in user code, it has no control over inode allocation; thus a full dump must be done to get a new set of directories reflecting the new inode numbering, even though the contents of the files are unchanged.

AUTHOR

restore and *rrestore* was developed by the University of California, Berkeley.

FILES

/dev/rmt/0m	default tape drive
/tmp/rstdr*	file containing directories on the tape
/tmp/rstmd*	owner, mode, and time stamps for directories
./restoresymtab	information passed between incremental restores

SEE ALSO

dump(1M), mkfs(1M), mount(1M), newfs(1M), rmt(1M).

NAME

revck - check internal revision numbers of HP-UX files

SYNOPSIS

/etc/revck *ref_files*

DESCRIPTION

revck checks the internal revision numbers of lists of files against reference lists. Each *ref_file* must contain a list of absolute path names (each beginning with **/**) and *whatstrings* (revision information strings from **what** — see *what*(1)). Path names begin in column 1 of a line, and have a colon appended to them. Each path name is followed by zero or more lines of *whatstrings*, one per line, each indented by at least one tab (this is the same format in which **what** outputs its results).

For each path name, **revck** checks that the file exists, and that executing **what** on the current path name produces results identical to the *whatstrings* in the reference file. Only the first 1024 bytes of *whatstrings* are checked.

ref_files are usually the absolute path names of the *revlist* files shipped with HP-UX. Each HP-UX software product includes a file named **/system/***product***/revlist** (for example, **/system/97070A/revlist**). The *revlist* file for each product is a reference list for the ordinary files shipped with the product, plus any empty directories on which the product depends.

FILES

/system/*product***/revlist** lists of HP-UX files and revision numbers

SEE ALSO

what(1).

DIAGNOSTICS

revck is silent except for reporting missing files or mismatches.

WARNINGS

revck produces unpredictable results if a *ref_file* is not in the right format.

NAME

rexd - RPC-based remote execution server

SYNOPSIS

/usr/etc/rpc.rexd [**-l** *log_file*] [**-m** *mountdir*] [**-r**]

DESCRIPTION

rexd is the RPC server for remote command execution. A **rexd** is started by **inetd** when a remote execution request is received (see *inetd*(1M)) **rexd** exits when command execution has completed.

If the user ID (uid) in the remote execution request is assigned to a user on the server, **rexd** executes the command as that user. If no user on the server is assigned to the uid, **rexd** does not execute the command. The **-r** option and **inetd.sec** security file allow for better access control (see *inetd.sec*(4)).

For noninteractive commands, standard output and error file descriptors are connected to sockets. Interactive commands use pseudo terminals for standard input, output and error (see *pty*(7)).

If the file system specified in the remote execution request is not already mounted on the server, **rexd** uses NFS to mount the file system for the duration of the command execution (see *nfs*(7)). **rexd** mounts file systems with the **nosuid** and **soft** options. For more details on mount options see *mount*(1M). If the server cannot mount the file system, an error message is returned to the client. By default, any mount points required by **rexd** are created below **/usr/spool/rexd**. To change the default location, use the **-m** option.

To configure **rexd**, the following entry must be present in the **/etc/inetd.conf** file (see *inetd.conf*(4)):

```
rpc stream tcp nowait root /usr/etc/rpc.rexd 100017 1 rpc.rexd options
```

Options

rexd recognizes the following options and command-line arguments:

-l *log_file*
: Log any diagnostic, warning, and error messages to *log_file*. If *log_file* exists, **rexd** appends messages to the file. If *log_file* does not exist, **rexd** creates it. Messages are not logged if the **-l** option is not specified.

  Information logged to the file includes date and time of the error, host name, process ID and name of the function generating the error, and the error message. Note that different RPC services can share a single log file because enough information is included to uniquely identify each error.

-m *mountdir*
: Create temporary mount points below directory *mountdir*. By default, **rexd** creates temporary mount points below **/usr/spool/rexd**. The directory *mountdir* should have read and execute permission for all users (mode 555). Otherwise, **rexd** denies execution for users that do not have read and execute permission.

-r
: Use increased security checking. When started with the **-r** option, **rexd** denies execution access to a client unless one of the following conditions is met:

  - The name of the client host is in **/etc/hosts.equiv** file on the server.
  - The user on the server that is associated with the uid sent by the client has an entry in **$HOME/.rhosts** specifying the client name on a line or the client name followed by at least one blank and the user's name.

    For example, assume a user whose login name is **mjk** is assigned to uid 7 on **NODE1** and executes the following **on** command:

    ```
    on NODE2 pwd
    ```

    User **mjk** on **NODE2** must have one of the following entries in **$HOME/.rhosts**:

    ```
    NODE1
    NODE1 mjk
    ```

DIAGNOSTICS

The following is a subset of the messages that could appear in the log file if the **-l** option is used. Some of

these messages are also returned to the client.

rexd: could not umount: *dir*
rexd was unable to **umount()** the user's current working file system. See WARNINGS for more details.

rexd: mountdir (*mountdir***) is not a directory**
The path name *mountdir*, under which temporary mount points are created, is not a directory or does not exist.

rexd: *command***: Command not found**
rexd could not find *command*.

rexd: *command***: Permission denied**
rexd was denied permission to execute *command*.

rexd: *command***: Text file busy**
The executable file is currently open for writing.

rexd: *command***: Can't execute**
rexd was unable to execute *command*.

rexd: root execution not allowed
rexd does not allow execution as user **root**.

rexd: User id*uid* **not valid**
The uid *uid* is not assigned to a user on the server.

rexd: User id*uid* **denied access**
rexd was started with the **-r** option and the remote execution request did not meet either of the conditions required by the **-r** option.

rexd: *host* **is not running a mount daemon**
The host *host* on which the user's current working directory is located is not running **mountd**. Therefore, **rexd** is unable to mount the required file system (see *mountd*(1M)).

rexd: not in export list for*file_system*
The host on which the client's current working directory is located does not have the server on the export list for file system *file_system* containing the client's current working directory. Therefore, **rexd** is unable to mount the required file system.

WARNINGS

The client's environment is simulated by **rexd**, but not completely recreated. The simulation of the client's environment consists of mounting the file system containing the client's current working directory (if it is not already mounted) and setting the user's environment variables on the server to be the same as the user's environment variables on the client. Therefore a command run by **rexd** does not always have the same effect as a command run locally on the client.

The **rex** protocol only identifies the client user by sending the uid of the client process and the host name of the client. Therefore, it is very difficult for **rexd** to perform user authentication. If a user on the server is assigned to the uid sent by the client, **rexd** executes the requested command as that user. If no user on the client is assigned to the uid sent by the client, **rexd** returns an error.

The **-r** option has been added to provide increased user authentication. However, the authentication provided is not foolproof, and is limited by the information passed by the **rex** protocol.

In order to simulate the client's environment, **rexd** mounts the file system containing the client's current working directory (if it is not already mounted). This mount is intended to be temporary for the duration of the command.

If **rexd** mounts a file system, it attempts to **umount()** the file system after the command has completed executing. However, if **rexd** receives a **SIGKILL** signal (see *signal*(2)), the file system is not unmounted. The file system remains mounted until the super-user executes the appropriate **umount** command or the server is rebooted.

rexd's attempt to umount the file system can also fail if the file system is busy. The file system is busy if it contains an open file or some user's current working directory. The file system remains mounted until the

super-user executes the appropriate **umount** command or the server is rebooted.

For more information on rex security issues, see *Using and Administering NFS Services*. Security issues and their consequences should be considered before configuring **rexd** to run on a system.

FILES

/dev/pty[pqr]*	master pseudo terminals
/dev/tty[pqr]*	slave pseudo terminals
/dev/ptym/pty[pqr]*	master pseudo terminals
/dev/pty/tty[pqr]*	slave pseudo terminals
/etc/inetd.conf	configuration file for *inetd*(1M)
/etc/hosts.equiv	list of equivalent hosts.
$HOME/.rhosts	user's private equivalence list.
/usr/spool/rexd/rexd*xxxxx*	temporary mount points for remote file systems where *xxxxx* is a string of alpha numeric characters.

AUTHOR

rexd was developed by Sun Microsystems, Inc.

SEE ALSO

on(1), inetd(1M), mount(1M), exports(4), inetd.conf(4), inetd.sec(4).

Installing and Administering NFS Services

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages

(Requires Optional ARPA Services Software)

NAME

rexecd - remote execution server

SYNOPSIS

/etc/rexecd [**-n**]

DESCRIPTION

rexecd is the server for the *rexec*(3N) routine; it expects to be started by the internet daemon (see *inetd*(1M)). *rexecd* provides remote execution facilities with authentication based on user account names and unencrypted passwords.

inetd(1M) calls *rexecd* when a service request is received at the port indicated for the "exec" service specification in **/etc/services**; see *services*(4). To run *rexecd*, the following line should be present in **/etc/inetd.conf**:

exec stream tcp nowait root /etc/rexecd rexecd

When a service request is received, the following protocol is initiated:

1. The server reads characters from the socket up to a null (**\0**) byte. The resultant string is interpreted as an ASCII number, base 10.
2. If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's host. If the first character sent is a null (**\0**), no secondary connection is made and the **stderr** of the command is sent to the primary stream. If the secondary connection has been made, *rexecd* interprets bytes it receives on that socket as signal numbers and passes them to the command as signals (see *signal*(2)).
3. A null-terminated user name of not more than 16 characters is retrieved on the initial socket.
4. A null-terminated, unencrypted, password of not more than 16 characters is retrieved on the initial socket.
5. A null-terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.
6. *rexecd* then validates the user as is done by *login*(1). If the authentication succeeds, *rexecd* changes to the user's home directory and establishes the user and group protections of the user. If any of these steps fail, *rexecd* returns a diagnostic message through the connection, then closes the connection.
7. A null byte is returned on the connection associated with **stderr** and the command line is passed to the normal login shell of the user with that shell's **-c** option. The shell inherits the network connections established by *rexecd*.

rexecd uses the following path when executing the specified command:

:/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin

Transport-level keepalive messages are enabled unless the **-n** option is present. The use of keepalive messages allows sessions to be timed out if the client crashes or becomes unreachable.

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

Username too long
The user name is longer than 16 characters.

Password too long
The password is longer than 16 characters.

Command too long
The command line passed exceeds the size of the argument list (as configured into the system).

Login incorrect
No password file entry for the user name existed or the wrong password was supplied.

No remote directory
The *chdir* command to the home directory failed.

No more processes
The server was unable to fork a process to handle the incoming connection.

Next step: Wait a period of time and try again. If the message persists, then the server's host may have a runaway process that is using all the entries in the process table.

shellname: ...
The user's login shell could not be started via *exec*(2) for the given reason.

WARNINGS
The password is sent unencrypted through the socket connection.

AUTHOR
rexecd was developed by the University of California, Berkeley.

SEE ALSO
remsh(1), inetd(1M), rexec(3N), inetd.conf(4), inetd.sec(4), services(4).

NAME

ripquery - query RIP gateways

SYNOPSIS

ripquery [-n] [-p] [-r] [-w *time*] *gateway* ...

DESCRIPTION

ripquery requests all routes known by a RIP gateway by sending a RIP request or POLL command. The routing information in any routing packets returned is displayed numerically and symbolically. **ripquery** should be used as a tool for debugging gateways, not for network management.

ripquery by default uses the RIP POLL command, which is an undocumented extension to the RIP specification supported by **routed** on SunOS 3.x and later, and by **gated** 1.4 and later. The RIP POLL command is preferred over the RIP REQUEST command because it is not subject to Split Horizon and/or Poisioned Reverse. See the RIP RFC for more information.

Options

-n	Normally network and host numbers are displayed both symbolically and numerically. When this option is specified, network and host numbers are only displayed numerically.
-p	Uses the RIP POLL command to request information from the routing table. This is the default, but is an undocumented extension supported only by some versions of SunOS 3.x and later versions of **gated**. If there is no response to the RIP POLL command, the RIP REQUEST command is tried.
-r	Uses the RIP REQUEST command to request information from the gateway's routing table. Unlike the RIP POLL command, all gateways should support the RIP REQUEST. If there is no response to the RIP REQUEST command, the RIP POLL command is tried.
-v	Version information about **ripquery** is displayed before querying the gateways.
-w	Specifies the time in seconds to wait for the initial response from a gateway. The default value is 5 seconds.

SEE ALSO

gated(1M).

RFC1058: Routing Information Protocol

(Requires Optional LAN/X.25 Software)

NAME

rlb - remote loopback diagnostic

SYNOPSIS

rlb [**-e**] [**-t**]

DESCRIPTION

rlb is a diagnostic program for LAN/9000 Local Area Network (LAN) products. to use this diagnostic, NetIPC software must be installed on the system.

rlb tests communication with other nodes on a LAN. The Remote Communications Test mode is used to initiate the exchange of messages between the local system and known remote nodes on the network. Messages can be exchanged with specified nodes or can automatically poll all nodes listed in the node name file. Parameters such as message length, number of messages, timeout value, and display of message round trip times can be set. *rlbdaemon* must be running on any system being interacted with when using Remote Communications Test mode.

Commands are read from **stdin**, prompts and error messages are written to **stderr**, and status and message exchange displays are written to **stdout**. The interrupt signal, usually the Break key, can be used to generate an interrupt of a currently executing command, if *rlb* is initiated interactively. If commands are read from a file, Ctrl-I causes control to return to the shell.

rlb accepts either complete command words or unique abbreviations of one or more letters. Commands treat uppercase and lowercase letters as equivalent. Multiple commands can be entered on one line if they are separated by spaces, tabs or commas.

The options are:

-e Echo the input commands on the output device.

-t Suppress the display of the command menu before each command prompt. This option is identically equivalent to the Test Selection mode **terse** command. The default for *rlb* is **verbose**.

Available Test Selection mode commands are:

menu Displays the Test Selection mode command menu.

quit Terminates the *rlb* program.

remote Puts the diagnostic in Remote Communications Test mode.

terse Suppresses display of command menus.

verbose Restores default display of command menus.

Commands available in the Remote Communications Test mode are:

name Prompts for the name of a node name file used by the **all** command to determine which nodes to exchange messages with. The node name file must be an ASCII text file and have the following format:

- One node name per line.
- Each node name can be followed by a comment.
- Each node name consists of the name field, and the optional environment and organization fields, which are separated by periods.
- Each field of the node name can be up to 16 alphanumeric characters in length and must begin with an alphabetic character.
- Empty node names are not allowed.

all

Causes *rlb* to poll all nodes on the local network listed in the node name file. The list of nodes to be polled is taken from the node name file. The node name file is specified with the Remote Communication Test mode **name** command. The default is **/etc/diagnodes**.

continue

Forces *rlb* to continue message exchanges with a remote node, even if transmit/receive message data

differs. The default is to terminate the exchange if the data differs.

display
Causes *rlb* to compute and display message round-trip times. Round-trip time is the time messages take to make the round trip between the local node and a remote node. The operator can enter a **trigger value** to control how often the times are displayed. The default for **display** is **off**.

end
Returns *rlb* to Test Selection mode.

length
Prompts for a new message length for messages being exchanged with remote nodes. The default length is 100 bytes and the maximum length is 1450 bytes.

menu
Displays the Remote Communications Test mode command menu.

number
Prompts for a new value for the number of messages to be exchanged with remote nodes. The default value is 1. To set **number** to a value greater than 10, you must have appropriate privileges. The maximum allowable number is $2^{31} - 1$.

quit
Terminates the *rlb* program.

single
Prompts for a node name, then executes message exchanges with that remote node.

timeout
Prompts for a new timeout value, which determines how long *rlb* waits for a response from a remote node. The default value is 10 seconds. The maximum allowable timeout value is 600 seconds.

AUTHOR
rlb was developed by HP.

SEE ALSO
linkloop(1M), nodename(1), ping(1M), rlbdaemon(1M).

NAME

rlbdaemon - remote loopback diagnostic server

SYNOPSIS

`/etc/rlbdaemon`

DESCRIPTION

rlbdaemon is the server that supports the remote-loopback diagnostic (**rlb**) client program. **rlb** and **rlbdaemon** together provide a diagnostic capability for LAN/9000 Local Area Network (LAN) products on HP 9000 computer systems.

To use this diagnostic, NetIPC software must be installed on the system. Only users with appropriate privileges can start the **rlbdaemon** server.

rlbdaemon waits for connections on a NetIPC call socket. When a connection arrives, **rlbdaemon** forks and invokes a server to exchange messages with the client process. The forked server process executes until the connection is closed. The parent **rlbdaemon** process continues to wait for another connection on the Netipc call socket. A maximum of 10 forked server processes can be executing at one time.

AUTHOR

rlbdaemon was developed by HP.

SEE ALSO

linkloop(1M), ping(1M), rlb(1M).

(Requires Optional ARPA Services Software)

NAME

rlogind - remote login server

SYNOPSIS

/etc/rlogind [**-ln**]

DESCRIPTION

rlogind is the server for the *rlogin*(1) program. It provides a remote login facility with authentication based on privileged port numbers. *rlogind* expects to be executed by the Internet daemon (*inetd*(1M)) when it receives a service request at the port indicated in the services database for **login** using the **tcp** protocol (see *services*(4)).

When a service request is received, the following protocol is initiated by *rlogind*:

1. *rlogind* checks the client's source port. If the port is not in the range 512 through 1023 (a "privileged port"), the server aborts the connection.
2. *rlogind* checks the client's source address and requests the corresponding host name (see *gethostent*(3N), *hosts*(4), and *named*(1M)). If it cannot determine the hostname, it uses the Internet dot-notation representation of the host address.

Once the source port and address have been checked, *rlogind* proceeds with the authentication process described in *hosts.equiv*(4). *rlogind* then allocates a pseudo-terminal (see *pty*(7)), and manipulates file descriptors so that the slave half of the pseudo-terminal becomes **stdin**, **stdout**, and **stderr** for a login process. The login process is an instance of *login*(1) invoked with the **-f** option if authentication has succeeded. If automatic authentication fails, *login*(1) prompts the user with the normal login sequence. The **-l** option to *rlogind* prevents any authentication based on the user's **.rhosts** file unless the user is logging in as super-user.

The *rlogind* process manipulates the master side of the pseudo-terminal, operating as an intermediary between the login process and the client instance of the *rlogin* program. The packet protocol described in *pty*(7) is used to enable and disable flow control via Ctrl-S/Ctrl-Q under the direction of the program running on the slave side of the pseudo-terminal, and to flush terminal output in response to interrupt signals. The login process sets the baud rate and **TERM** environment variable to correspond to the client's baud rate and terminal type (see *environ*(5)).

Transport-level keepalive messages are enabled unless the **-n** option is present. The use of keepalive messages allows sessions to be timed out if the client crashes or becomes unreachable.

To start *rlogind* from the Internet daemon, the configuration file **/etc/inetd.conf** must contain an entry as follows:

login stream tcp nowait root /etc/rlogind rlogind

DIAGNOSTICS

Errors in establishing a connection cause an error message to be returned with a leading byte of 1 through the socket connection, after which the network connection is closed. Any errors generated by the login process or its descendents are passed through by the server as normal communication.

fork: No more processes

The server was unable to fork a process to handle the incoming connection.

Next step: Wait a period of time and try again. If this message persists, the server's host may have runaway processes that are using all the entries in the process table.

Cannot allocate pty on remote host

The server was unable to obtain a pseudo-terminal for use with the login process. Either all pseudo-terminals were in use, or the pty driver has not been properly set up (see *pty*(7)).

Next step: Check the pty configuration of the host where *rlogind* executes.

Permission denied

The server denied access because the client was not using a reserved port. This should only happen to interlopers trying to break into the system.

/bin/login: ...

The login program could not be started via *exec*(2) for the reason indicated.

Next step: Try to correct the condition causing the problem. If this message persists, contact your system administrator.

WARNINGS

The "privileged port" authentication procedure used here assumes the integrity of each host and the connecting medium. This is insecure, but is useful in an "open" environment. Note that any passwords are sent unencrypted through the socket connection.

AUTHOR

rlogind was developed by the University of California, Berkeley.

FILES

/etc/hosts.equiv	list of equivalent hosts
$HOME/.rhosts	user's private equivalence list

SEE ALSO

login(1), rlogin(1), inetd(1M), named(1M), gethostent(3N), ruserok(3N), hosts(4), hosts.equiv(4), inetd.conf(4), services(4), environ(5), pty(7).

NAME

rlp - send LP line printer request to a remote system

SYNOPSIS

/usr/lib/rlp -I *id* [**-C** *class*] [**-J** *job*] [**-T** *title*] [**-i**[*numcols*]] [*-kfont*] [**-w** *num*] [**-cdfghlnptv**] *file*

DESCRIPTION

rlp transfers a spooling request to a remote system to be printed. *rlp* communicates with a spooling daemon on a remote system to transfer the spooling request. Options can be set only on the original system. Transfers of a remote request use only the **-I** option and the file.

This command is intended to be used only by the spool system in response to the *lp*(1) command and should not be invoked directly.

Options

-I*id* The argument *id* is the request ID.

-C *class* Take the following argument as a job classification for use on the banner page.

-J *job* Take the following argument as the job name to print on the banner page. Normally, the first file's name is used.

-T *title* Use the next argument as the title used by *pr*(1) instead of the file name. **-T** is ignored unless the **-p** option is specified.

-h Suppress the printing of the banner page.

-i[*numcols*] Cause the output to be indented. If the next argument is numeric, it is used as the number of blanks to be printed before each line; otherwise, 8 characters are printed.

-kfont Specify a *font* to be mounted on font position *k*, where *k* is from **1** through **4**.

-w*num* Use the immediately following number as the page width for *pr*(1).

The following single-letter options are used to notify the line printer spooler that the files are not standard text files. The spooling system uses the appropriate filters (if the option is supported) to print the data accordingly. These options are mutually exclusive.

-c The files are assumed to contain data produced by *cifplot*.

-d The files are assumed to contain data from *tex* (DVI format).

-f Use a filter that interprets the first character of each line as a standard FORTRAN carriage control character.

-g The files are assumed to contain standard plot data as produced by the *plot* routines.

-l Use a filter that suppresses page breaks.

-n The files are assumed to contain data from *ditroff* (device independent *troff*).

-p Use *pr*(1) to format the files.

-t The files are assumed to contain data from *troff* (cat phototypesetter commands).

-v The files are assumed to contain a raster image for devices such as the Benson Varian.

HP Clustered Environment

In the HP Clustered Environment, all spooling is handled as if the cluster nodes were a single system and all printers attached to either the cluster server or clients can be available. Remote spooling applies to spooling from or to machines outside of the cluster nodes.

WARNINGS

Some remote line printer models may not support all of these options. Options not supported are silently ignored.

When *rlp* is transferring a request that originated on another system, only the **-I** option and the file is used. This saves *rlp* from having to set the various options multiple times. Specifying unused options does not produce an error.

AUTHOR

rlp was developed by the University of California, Berkeley and HP.

FILES

/etc/passwd
/usr/lib/rlpdaemon
/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1), rcancel(1M), rlpdaemon(1M), rlpstat(1M).

NAME

rlpdaemon - remote spooling line printer daemon, message write daemon

SYNOPSIS

/usr/lib/rlpdaemon [**-i**] [**-l**] [**-L** *logfile*]

DESCRIPTION

rlpdaemon is a line printer daemon (spool area handler) for remote spool requests. *rlpdaemon* is normally invoked at boot time from the **/etc/rc** file or started by *inetd*(1M), when necessary. *rlpdaemon* runs on a system that receives requests to be printed. *rlpdaemon* transfers files to the spooling area, displays the queue, or removes jobs from the queue.

rlpdaemon is also used as a server process to write a message on the user's terminal, upon receiving a request from a remote system.

Options

-i Prevent *rlpdaemon* from remaining after a request is processed. This is required if *rlpdaemon* is started from *inetd*(1M).

-l Cause *rlpdaemon* to log error messages and valid requests received from the network to the file **/usr/spool/lp/lpd.log**. This can be useful for debugging.

-L *logfile* Change the file used for writing error conditions from the file **/usr/spool/lp/lpd.log** to *logfile*.

When *rlpdaemon* is started by *inetd*(1M), access control is provided via the file **/usr/adm/inetd.sec** to allow or prevent a host from making requests. When *rlpdaemon* is not started by *inetd*(1M), all requests must come from one of the machines listed in the file **/etc/hosts.equiv** or **/usr/spool/lp/.rhosts**. When **/usr/spool/lp/.rhosts** is used for access, the user name should be **lp**.

The following entry should exist in **/etc/services** for remote spooling:

```
printer         515/tcp         spooler
```

HP Clustered Environment

In the HP Clustered Environment, all spooling is handled as if the cluster nodes were a single system and all printers attached to either the cluster server or clients can be available. Remote spooling applies to spooling from or to machines outside of the cluster nodes.

EXAMPLES

To start *rlpdaemon* from **/etc/rc**, invoke the command:

/usr/lib/rlpdaemon

To start *rlpdaemon* from *inetd,* the following line should be included in the file **/etc/inetd.conf**:

printer stream tcp nowait root /usr/lib/rlpdaemon rlpdaemon -i

WARNINGS

If the remote system is the same as the local system and *rlpdaemon* was not started by *inetd*(1M), the local system name *must* be included in file **/etc/hosts.equiv**.

AUTHOR

rlpdaemon was developed by the University of California, Berkeley and HP.

FILES

/etc/hosts.equiv
/etc/services
/usr/spool/lp/*
/usr/adm/inetd.sec

SEE ALSO

accept(1M), enable(1), lp(1), inetd(1M), lpadmin(1M), lpsched(1M), lpstat(1), rcancel(1M), rlp(1M), rlpdaemon(1M), rlpstat(1M). hosts.equiv(4), inetd.conf(4), inetd.sec(4), services(4).
Installing and Administering ARPA Services,
HP-UX System Administrator manuals.

NAME

rlpstat - print status of LP spooler requests on a remote system

SYNOPSIS

`/usr/lib/rlpstat` [`-d`*printer*] [`-u` *user*] [*id* ...]

DESCRIPTION

`rlpstat` reports the status of the specified jobs or all requests associated with a user. If no arguments are specified, `rlpstat` reports on any requests currently in the queue.

For each request submitted (i.e., each invocation of `lp` — see *lp*(1)) `rlpstat` reports the request ID, user's name, total size of the request, date of the request, and, if it is being transferred, the device.

This command is intended to be used only by the spool system in response to the `lpstat` command and should not be invoked directly (see *lpstat*(1M)).

Options

`rlpstat` recognizes the following options and command-line arguments:

`-d` *printer*	Specify a particular printer. Otherwise, the default line printer is used (or the value of the `LPDEST` environment variable).
`-u` *user*	Status is requested on all requests for the user who executed the `rlpstat` command on the specified printer (see the `-d` option).
id	Status is requested on the specified request IDs (as returned by `lp`). All the request IDs must be for the same printer.

DEPENDENCIES

HP Clustered Environment

In the HP Clustered Environment, all spooling is handled as if the cluster nodes were a single system and all printers attached to either the cluster server or clients can be available. Remote spooling applies to spooling from or to machines outside of the cluster nodes.

AUTHOR

`rlpstat` was developed by the University of California, Berkeley, and HP.

FILES

`/usr/spool/lp/*`

SEE ALSO

enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1), rcancel(1M), rlp(1M), rlpdaemon(1M).

NAME

rmfn - remove HP-UX functionality (partitions and filesets)

SYNOPSIS

`/etc/rmfn` [`-y`] [`-s` *series*] [`-l` *partition_or_fileset_list*] [`-o` *pre-Release_8.0_fileset_list*] [`-f` *file*]

DESCRIPTION

`rmfn` is used to interactively or non-interactively remove functionality from a system in units of *filesets* or *partitions*.

If no options are specified, `rmfn` runs interactively, providing help screens to aid in selecting `rmfn` soft-key options. If one or more options are specified, `rmfn` runs non-interactively as specified by the options given.

`rmfn` can only be executed by the super-user. It can be used while the system is running in multi-user mode, but only one instance of `rmfn` can be running on the system at a time.

In an HP Clustered environment, `rmfn` can only be run on the cluster server.

`rmfn` understands dependencies between filesets. It does not remove a fileset if it contains functions required by other filesets on the system that have not been selected for removal or cannot be removed.

During the removal process, if a file cannot be removed because it is busy (**ETXTBSY**), `rmfn` moves the file to #*file* and the user should remove this #*file* later.

Options

`rmfn` supports the following options when invoked non-interactively:

`-l` *partition_or_fileset_list*
Remove the partitions and/or filesets listed in the partition/fileset list. If two or more partitions and/or filesets are specified for removal, the entire list must be enclosed within a pair of double quotes ("). Each item in the list should have one of the following formats:

partition`.`*fileset*	removes a fileset within a partition
partition	removes all the filesets in a partition
fileset	removes an individual fileset

`-f` *file*
Read the list of filesets or partitions to be removed from the specified file, rather than from the command line (see `-l` option for the list format).

Blank lines and comments (lines beginning with #) in *file* are ignored.

`-y`
Give `rmfn` permission to remove filesets that depend on filesets listed in the `-l` or `-f` options. `rmfn` removes filesets specified in the `-l` or `-f` options and all filesets that depend on them. If `-y` is not specified, `rmfn` checks dependencies for each fileset listed in the `-l` or `-f` options, and does not remove the fileset if it is required by any other filesets on the system that either are not also listed for removal or cannot be removed.

`-s` *series*
Specify the system type (`300` (for Series 300 or 400), `700`, or `800`) for the filesets listed in the `-l` or `-f` options. If `-s` is not specified, the default is the type of the system on which `rmfn` is run.

`-o` *fileset_list*
Remove pre-Release-8.0 filesets specified in the fileset list. Filesets prior to HP-UX Release 8.0 did not contain any dependency information. To accommodate filesets that may have been installed prior to 8.0, this option prevents `rmfn` from checking dependencies on pre-8.0 filesets before removal. However, `rmfn` is not allowed to remove fileset `UX_CORE` and fileset `TOOL`. If two or more partitions and/or filesets are specified for removal, the entire list must be enclosed within a pair of double quotes ("). The format is:

fileset	remove an individual fileset
`\*`	remove all pre-8.0 filesets

NOTE: The * must be escaped in the command line to prevent its expansion by the shell.

Mounted Volumes and Symbolic Links

rmfn does not remove files under directories on read-only file systems or remotely mounted file systems. **rmfn** requires that all the disks listed in **/etc/checklist** be mounted (disks must be listed in **/etc/mnttab**).

rmfn does not produce any error messages if an attempt is made to remove remote files.

rmfn does not follow symbolic links. If an entry in an **/etc/filesets** file is a symbolic link, **rmfn** removes the symbolic link but leaves the link target intact. If an entry in the fileset file is an ordinary file or a hard link, the file or link is removed but any symbolic links to the file or hard link, if they exist, are left unresolved.

HP Clustered Environment

In a heterogeneous cluster (containing systems of dissimilar architecture), when removing a fileset from the server for a given architecture, if the fileset contains files that are also used by the other architecture but the corresponding fileset for the other architecture is being left on the system, then **rmfn** does not remove those shared files.

rmfn removes empty context-dependent file (CDF) directories.

Kernel Reconfiguration

If a removed fileset contains a piece of the kernel, a message warning the user to reconfigure the kernel is displayed and also recorded in the log file. However, **rmfn** does not modify any of the kernel generation files.

Recovering Removed Functionality

To recover functionality that was previously removed, use **update** (see *update*(1M)).

Index File

Any fileset being removed by **rmfn** *must* have entries in the **/system** and **/etc/filesets** directories. A fileset whose **index** file resides in directory **/system/***fileset* can be removed interactively or non-interactively by specifying the **-l** or **-f** option in the command line. A fileset that does not have an **index** file in directory **/system/***fileset* is treated as a pre-Release-8.0 fileset. It can be removed by using the **-o** command line option for non-interactive execution.

RETURN VALUE

rmfn returns the following values:

- **0** **rmfn** ran successfully to completion.
- **1** an error occurred and no files were removed.
- **2** an error occurred and some filesets might have been removed; review the log file for details (see FILES below)

DIAGNOSTICS

Messages displayed during interactive execution and error messages resulting from invalid non-interactive invocation are self-explanatory. For information about any failures encountered while removing filesets, inspect the log file (see FILES below).

EXAMPLES

Remove filesets **ARABIC** and **CHINESE** from the system.

```
rmfn -l ARABIC CHINESE
```

Remove all Series 300 system filesets listed in file **/tmp/remove.in**.

```
rmfn -s 300 -f  /tmp/remove.in
```

Remove Series 800 system fileset **NLIO-MIN** and all other filesets on the system that depend on it.

```
rmfn -s 800 -l NLIO-MIN -y
```

Remove Release 7.0 fileset **FORTRAN** from the system.

```
rmfn -o FORTRAN
```

Remove all Series 300 Release 7.0 filesets on the system.

```
rmfn -o \* -s 300
```

WARNINGS

Always review the log file after a removal for relevant warnings, errors, and messages.

In the interactive mode, **rmfn** displays the size of each fileset on the system. **rmfn** also reports the amount of disk space freed after each fileset has been removed, and the total amount of disk space freed when the removal is completed. The size information is approximate.

Do not press the **Print/Enter** key and the **Extend char** key while running **rmfn**. Pressing these keys may cause unexpected results.

rmfn removes files, hard links, and symbolic links as they are encountered in the fileset file in **/etc/filesets**. However, if a fileset file created by an independent (non-HP) party contains the names of one or more directories, the behavior of **rmfn** with respect to those directories is undefined.

Do not modify the **/etc/filesets/***fileset* file at any time, and do not create, modify, or remove any files in directory **/system/***fileset*. **rmfn** extracts fileset information from **/etc/filesets/***fileset* files and uses the files in **/system/***fileset* to ensure correct fileset removal.

AUTHOR

rmfn was developed by HP.

FILES

/etc/rmfn	executable file
/tmp/rmfn.log	log file describing the events that occurred during the **rmfn** process, including errors, warnings, notes, and messages
/etc/checklist	list of volumes that should be mounted
/etc/mnttab	list of volumes currently mounted
/etc/filesets/*fileset*	files containing list of files for each fileset
/system/*fileset*	directories containing files related to each fileset

SEE ALSO

mount(1M), regen(1M), update(1M).
HP-UX System Administrator manuals.

NAME

rmsf - remove a special file

SYNOPSIS

/etc/rmsf [**-a**|**-k**] *special_file* ...
/etc/rmsf [**-k**] [**-d** *driver* | **-C** *class*] **-H** *hw_path*

DESCRIPTION

rmsf removes one or more special files from the current directory, and potentially removes information about the associated device or devices from the system.

If a *special_file* is specified, without any options, **rmsf** removes that single special file from the current directory; the definition of the device remains in the system. If the **-k** option is specified, the definition of the device is removed from the system without removing the special file. If the **-a** option is specified, the definition of the device is removed from the system along with all special files in the current directory that map to it.

If a *hw_path* is specified alone, all special files in the current directory mapping to devices at that hardware path and the system definition of those devices are removed. The **-C** and **-d** options remove only those special files in the current directory that are associated with the given device driver or that belong to the given device class, respectively. This is useful when there is more than one type of special file mapped to a single hardware path. If the **-k** option is specified, the definition of all devices at that hardware path are removed from the system, again without removing any special file or files.

Options

rmsf recognizes the following options and command-line options:

-C *class* — Match devices belonging to the specified device *class*. Device classes are defined in file **/etc/master**, and can be listed using *lsdev*(1M). Cannot be used with **-d**.

-a — Remove the definition of the device from the system along with all special files in the current directory that refer to the device. Cannot be used with **-k**.

-d *driver* — Match devices that are controlled by the specified device *driver*. Cannot be used with **-C**.

-k — Remove the definition of the device from the system, but not any special files. Cannot be used with **-a**.

-H *hw_path* — A hardware path specifies the addresses of the hardware components leading to a device. It consists of a string of numbers each suffixed by a slash (/), followed by a string of numbers separated by periods (.). Hardware components suffixed by slashes indicate bus converters and may not be necessary on some systems. Hardware components suffixed by . indicate the addresses of the remaining hardware components on the path to the device.

RETURN VALUE

rmsf returns **0** upon normal completion and **1** if an error occurred.

DIAGNOSTICS

Most of the diagnostic messages from **rmsf** are self explanatory. Listed below are some messages deserving further clarification. Errors cause **rmsf** to halt immediately; warnings allow the program to continue.

Errors

No such device in the system
No device in the system matched the options specified. Use **ioscan** to list the devices in the system (see *ioscan*(1M)).

special_file **is not a special file**
The file is not associated with an I/O device.

Warnings

Cannot remove *driver* **at** *hw_path*
The definition of the device located at *hw path* and controlled by *driver* cannot be removed from the kernel.

No device associated with *special_file*
The special file does not map to a device in the system; the file is removed unless the **-k** option was specified.

EXAMPLES

Remove the special file **mux0** from the current directory:

```
rmsf mux0
```

Remove the system definition of the device associated with **/dev/lp0** along with all special files that refer to the device:

```
rmsf -a /dev/lp0
```

Remove the system definitions for all devices associated with hardware path 8.4.0:

```
rmsf -k 8.4.0
```

AUTHOR

rmsf was developed by HP.

FILES

/dev/config
/etc/ioconfig
/etc/master

SEE ALSO

rm(1), insf(1M), ioscan(1M), lssf(1M), mksf(1M), ioconfig(4).

NAME

rmt - remote magnetic-tape protocol module

SYNOPSIS

`/etc/rmt`

DESCRIPTION

`rmt` is a program used by the remote dump and restore programs for manipulating a magnetic tape drive through an interprocess communication (IPC) connection. The `fbackup` and `frecover` commands also use `rmt` to achieve remote backup capability (see *fbackup*(1M) and *frecover*(1M)). `rmt` is normally started up with an `rexec()` or `rcmd()` call (see *rexec*(3C) and *rcmd*(3C)).

`rmt` accepts requests specific to the manipulation of magnetic tapes, performs the commands, then responds with a status indication. DDS devices that emulate magnetic tapes are also supported. All responses are in ASCII and in one of two forms. Successful commands have responses of

A*number*`\n`

where *number* is an ASCII representation of a decimal number. Unsuccessful commands are responded to with

E*error-number*`\n`*error-message*`\n`

where *error-number* is one of the possible error numbers described in *errno*(2) and *error-message* is the corresponding error string as printed from a call to `perror()` (see *perror*(3C)). The protocol is comprised of the following commands (a space is present between each token):

`O` *device mode*	Open the specified *device* using the indicated *mode*. *device* is a full pathname and *mode* is an ASCII representation of a decimal number suitable for passing to `open()` (see *open*(2)). If a device is already open, it is closed before a new open is performed.
`o` *device mode*	Open the specified *device* using the indicated *mode*. *device* is a full pathname and *mode* is an ASCII representation of an octal number suitable for passing to `open()`. If a device is already open, it is closed before a new open is performed.
`C` *device*	Close the currently open device. The *device* specified is ignored.
`L` *whence offset*	Perform an `lseek()` operation using the specified parameters (see *lseek*(2)). The response value is that returned from by `lseek()`.
`W` *count*	Write data onto the open device. `rmt` reads *count* bytes from the connection, aborting if a premature end-of-file is encountered. The response value is that returned from by `write()` (see *write*(2)).
`R` *count*	Read *count* bytes of data from the open device. If *count* exceeds the size of the data buffer (10 Kbytes), it is truncated to the data buffer size. `rmt` then performs the requested `read()` and responds with A*count-read*`\n` if the read was successful. Otherwise an error is returned in the standard format. If the read was successful, the data read is then sent.
`I` *operation count*	Perform a `MTIOCOP ioctl()` command using the specified parameters. Parameters are interpreted as ASCII representations of the decimal values to be placed in the `mt_op` and `mt_count` fields of the structure used in the `ioctl()` call. The return value is the *count* parameter when the operation is successful.
`S`	Return the status of the open device, as obtained with a `MTIOCGET ioctl()` call. If the operation was successful, an ACK is sent with the size of the status buffer, then the status buffer is sent (in binary).
`s`	Return the status of the open device, as obtained with a `fstat()` call. If the operation was successful, an ACK is sent with the size of the status buffer, then the status buffer is sent (in binary). `f` Return the status of the open device, as obtained with a `fstat()` call. If the operation was successful, an ACK is sent with the size of the status buffer, then the status buffer is sent in the following ASCII format:

machine<blank>*value*<newline>
stat_struct_member_name<blank>*value*<newline>

The end of the data is indicated by an ASCII NULL character. See `/usr/include/sys/stat.h` for the `struct stat` definition. In addition to the struct stat information, there is an entry in the buffer describing the machine type as returned from a `uname()` call (see *uname*(2)). In the above format "machine" is a key word. All fields except `st_spare4` of the `struct stat` are returned.

m
Return the status of the open device, as obtained with a `MTIOCGET ioctl()` call. If the operation was successful, an ack is sent with the size of the status buffer, then the status buffer is sent in the following ASCII format:

machine<blank>*value*<newline>
mtget_struct_member_name<blank>*value*<newline>

The end of the data is indicated by an ASCII NULL character. See `/usr/include/sys/mtio.h` for the `struct mtget` definition. In addition to the struct mtget information there is an entry in the buffer describing the machine type as returned from a `uname()` call. In the above format "machine" is a keyword.

Any other command causes `rmt` to exit.

DIAGNOSTICS

All responses are of the form described above.

DEPENDENCIES

Series 300/400

The device status is returned in the fields `mt_dsreg1` and `mt_dsreg2`. `mt_dsreg1` contains the following information in the least significant byte:

Bit	7(MSB)	6	5	4	3	2	1	0(LSB)
Status	EOF	BOT	EOT	RecErr	CmdRej	WriteP	UnErr	Online

Online	online status
UnErr	unrecognized error
WriteP	write protect
CmdRej	command reject
RecErr	recovered error
EOT	beyond end of tape
BOT	at load point
EOF	at end of file

The contents of the least significant byte of the `mt_dsreg2` are:

Bit	7(MSB)	6	5	4	3	2	1	0(LSB)
Status	6250	UnkDen	ParErr	TimErr	Run	Door	Long	ImRes

ImRes	immediate response mode
Long	long records supported
Door	door open
Run	tape runaway
TimErr	data timing error
ParErr	data parity error
UnkDen	unknown density
6250	6250 GCR format

Series 700/800

The device status is returned in the field `mt_gstat`. `/usr/include/sys/mtio.h` contains defined macros for checking the status bits.

AUTHOR

`rmt` was developed by the University of California, Berkeley.

SEE ALSO

ftio(1), fbackup(1M), frecover(1M), dump(1M), restore(1M), rcmd(3C), rexec(3C).

WARNINGS

Use of this command for remote file access protocol is discouraged.

NAME

route - manually manipulate the routing tables

SYNOPSIS

/etc/route [**-f**] [**-n**] **add** [**net** | **host**] *destination gateway* [*count*]
/etc/route [**-f**] [**-n**] **delete** [**net** | **host**] *destination gateway* [*count*]
/etc/route -f [**-n**]

route is used to manipulate the network routing tables manually, and accessible only by users who have appropriate privileges. **route** supports two commands:

add Add a route.

delete delete a route.

When adding a route, if the route already exists, a message is printed and nothing changes.

Other command line arguments are:

net or **host** specifies the type of *destination* address. If not specified, routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. If the *destination* has a "local address part" of **INADDR_ANY(0)**, the route is assumed to be to a network; otherwise, it is treated as a route to a host.

destination destination host system where the packets will be routed. *destination* can be either a host name (the official name or an alias, see *gethostbyname*(3N)), a network name (the official name or an alias, see *getnetbyname*(3N)), an Internet address in "dot" notation (see *inet*(3N)), or the keyword **default**, which signifies the wildcard gateway route (see *routing*(7)).

gateway The gateway through which the destination is reached. *gateway* can be either a host name (the official name or an alias, see *gethostbyname*(3N)), or an Internet address in "dot" notation.

count An integer that indicates whether the gateway is a remote host or the local host. If the route leads to a destination via a remote gateway, *count* should be a number greater than 0. If the route leads to *destination* and the gateway is the local host, *count* should be 0. The default for *count* is zero. The result is not defined if *count* is negative.

All symbolic names specified for a *destination* or *gateway* are looked up first as a hostname using **gethostbyname()**; if the hostname is not found, the *destination* is searched as a network name using **getnetbyname()**. *destination* and *gateway* can be in dot notation (see *inet*(3N)). If the **-n** option is not specified, any host and network addresses are displayed symbolically according to the name returned by **gethostbyaddr()** and **getnetbyaddr()**, respectively, except for the default network address (printed as **default**) and addresses that have unknown names. Addresses with unknown names are printed in Internet dot notation (see *inet*(3N) for more information regarding this format). If the **-n** option is specified, any host and network addresses are printed in Internet dot notation except for the default network address which is printed as **default**.

If the **-f** option is specified, **route** deletes all route table entries that specify a remote host for a gateway. If this is used with one of the commands described above, the entries are deleted before the command's application.

Output

add *destination*: **gateway** *gateway* **flags** *flags*
The specified route is being added to the tables.

delete *destination*: **gateway** *gateway* **flags** *flags*
The specified route is being deleted from the tables.

Flags

The following truth table can be used to help understand the relationship between count, destination type, flags, and route type.

Count	Destination Type	Flags	Route Type
=0	network	1=U	route to a network via a gateway which is the local host itself
>0	network	3=UG	route to a network via a gateway which is a remote host
=0	host	5=UH	route to a host via a gateway which is the local host itself
>0	host	7=UGH	route to a host via a gateway which is a remote host
=0	"default"	1=U	wildcard route via the local host
>0	"default"	3=UG	wildcard route via a remote gateway

DIAGNOSTICS

`delete a route that does not exist`
The specified route was not in the route table.

`add a route that already exists`
The specified entry is already in the route table.

`add too many routes`
The routing table is full.

WARNINGS

Reciprocal **`route`** commands must be executed on the local host, the destination host, and all intermediate hosts if routing is to succeed in the cases of virtual circuit connections or bidirectional datagram transfers.

DEPENDENCIES

The HP-UX implementation of **`route`** does not presently support a **`change`** command argument.

AUTHOR

`route` was developed by the University of California, Berkeley.

FILES

`/etc/networks`
`/etc/hosts`

SEE ALSO

netstat(1), ifconfig(1M), inet(3N), gethostbyname(3N), gethostbyaddr(3N), getnetbyname(3N), getnetbyaddr(3N), routing(7).

NAME

rpcinfo - report RPC information

SYNOPSIS

```
/usr/etc/rpcinfo -p [host]
/usr/etc/rpcinfo [-n portnum] -u host program [version]
/usr/etc/rpcinfo [-n portnum] -t host program [version]
/usr/etc/rpcinfo -b program version
/usr/etc/rpcinfo -d program version
```

DESCRIPTION

`rpcinfo` makes an RPC call to an RPC server and reports what it finds.

Options

`rpcinfo recognizes the following command-line options`

`-p` *host* — Probe the portmapper on *host* and print a list of all registered RPC programs. If *host* is not specified, it defaults to the value returned by `hostname` (see *hostname*(1)).

`-n` *portnum* — Use *portnum* as the port number for the `-t` and `-u` options instead of the port number given by the portmapper.

`-u` — Make an RPC call to procedure 0 of *program* on the specified *host* using UDP and report whether a response was received.

`-t` — Make an RPC call to procedure 0 of *program* on the specified *host* using TCP and report whether a response was received.

`-b` — Make an RPC broadcast to procedure 0 of the specified *program* and *version* using UDP and report all hosts that respond.

`-d` — Delete registration for the RPC service of the specified *program* and *version*. Only users with appropriate privileges can use this option.

program — Can be either a name or a number.

version — If specified, `rpcinfo` attempts to call that version of the specified *program*. Otherwise, `rpcinfo` attempts to find all the registered version numbers for the specified program by calling version 0, then attempts to call each registered version. (Version 0 is presumed to not exist, but if version 0 does exist, `rpcinfo` attempts to obtain the version number information by calling an extremely high version number instead.) Note that *version* must be specifed when the `-b` and `-d` options are used.

EXAMPLES

Show all of the RPC services registered on the local machine:

```
rpcinfo -p
```

Show all of the RPC services registered on the machine named `klaxon`:

```
rpcinfo -p klaxon
```

Show all machines on the local net that are running the Network Information Service (NIS):

```
rpcinfo -b ypserv 1 | sort | uniq
```

where `1` is the current NIS *version* obtained from the results of the `-p` option in the previous example.

Delete the registration for version 1 of the `walld` service:

```
rpcinfo -d walld 1
```

[Note that `walld` is the RPC program name for `rwalld` (see *rwalld*(1m))].

WARNINGS

In releases prior to Sun UNIX 3.0, the Network File System (NFS) did not register itself with the portmapper; *rpcinfo* cannot be used to make RPC calls to the NFS server on hosts running such releases. Note that this does not apply to any HP releases of NFS.

AUTHOR

rpcinfo was developed by Sun Microsystems, Inc.

FILES
/etc/rpc names for RPC program numbers

SEE ALSO
rpc(4), portmap(1M),
Programming and Protocols for NFS Services.

INTERNATIONAL SUPPORT
8-bit data, 16-bit data, messages

NAME

rquotad - remote quota server

SYNOPSIS

`/usr/etc/rpc.rquotad`

DESCRIPTION

`rquotad` is an RPC server that returns quotas for a user of a local file system currently mounted by a remote machine by means of NFS (see *rpc*(3C)). The results are used by `quota` to display user quotas for remote file systems (see *quota*(1)). `rquotad` is normally invoked by `inetd` (see *inetd*(1M)).

AUTHOR

Disk Quotas were developed by the University of California, Berkeley, Sun Microsystems, Inc., and HP.

FILES

`/quotas` quota file at the file system root

SEE ALSO

inetd(1M), rpc(3C), services(4), quota(5), nfs(7).

NAME

rstatd - kernel statistics server

SYNOPSIS

/usr/etc/rpc.rstatd [**-l** *log_file*] [**-e**|**-n**]

DESCRIPTION

rstatd is an RPC server that returns performance statistics obtained from the kernel. The **rup** utility prints this information (see *rup*(1)).

inetd invokes **rstatd** through **/etc/inetd.conf** (see *inetd*(1M)).

Options

rstatd recognizes the following options and command-line arguments:

-l *log_file* Log any errors to the named log file, *log_file*. Errors are not logged if the **-l** option is not specified.

Information logged to the file includes date and time of the error, the host name, process ID and name of the function generating the error, and the error message. Note that different services can share a single log file because enough information is included to uniquely identify each error.

-e Exit after serving each RPC request. Using the **-e** option, the **inetd** security file **/usr/adm/inetd.sec** can control access to RPC services.

-n Exit only if

- **portmap** dies (see *portmap*(1M)),
- another **rpc.rstatd** registers with **portmap**, or
- **rpc.rstatd** becomes unregistered with *portmap*.

The **-n** option is more efficient since a new process is not launched for each RPC request. Note, this option is the default.

AUTHOR

rstatd was developed by Sun Microsystems, Inc.

SEE ALSO

rup(1), inetd(1M), portmap(1M), inetd.conf(4), inetd.sec(4), services(4).

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages.

NAME

runacct - run daily accounting

SYNOPSIS

/usr/lib/acct/runacct [*mmdd* [*state*]]

DESCRIPTION

runacct is the main daily accounting shell procedure. It is normally initiated via *cron*(1M). *runacct* processes connect, fee, disk, and process accounting files. It also prepares summary files for *prdaily* or billing purposes.

runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into **active**. When an error is detected, a message is written to **/dev/console**, mail (see *mail*(1), *mailx*(1), or *elm*(1)) is sent to **root** and **adm**, and *runacct* terminates. *runacct* uses a series of lock files to protect against re-invocation. The files **lock** and **lock1** are used to prevent simultaneous invocation, and **lastdate** is used to prevent more than one invocation per day.

runacct breaks its processing into separate, restartable *states* using **statefile** to remember the last *state* completed. It accomplishes this by writing the *state* name into **statefile**. *runacct* then looks in **statefile** to see what it has done and to determine what to process next. *states* are executed in the following order:

SETUP	Move active accounting files into working files.
WTMPFIX	Verify integrity of **wtmp** file, correcting date changes if necessary.
CONNECT1	Produce connect session records in **ctmp.h** format.
CONNECT2	Convert **ctmp.h** records into **tacct.h** format.
PROCESS	Convert process accounting records into **tacct.h** format.
MERGE	Merge the connect and process accounting records.
FEES	Convert output of *chargefee* into **tacct.h** format and merge with connect and process accounting records.
DISK	Merge disk accounting records with connect, process, and fee accounting records.
MERGETACCT	Merge the daily total accounting records in **daytacct** with the summary total accounting records in **/usr/adm/acct/sum/tacct**.
CMS	Produce command summaries.
USEREXIT	Any installation-dependent accounting programs can be included here.
CLEANUP	Cleanup temporary files and exit.

To restart *runacct* after a failure, first check the **active** file for diagnostics, then fix up any corrupted data files such as **pacct** or **wtmp**. The **lock** files and **lastdate** file must be removed before *runacct* can be restarted. The argument *mmdd* is necessary if *runacct* is being restarted, and specifies the month and day for which *runacct* will rerun the accounting. Entry point for processing is based on the contents of **statefile**; to override this, include the desired *state* on the command line to designate where processing should begin.

EXAMPLES

To start *runacct*.

nohup runacct 2> /usr/adm/acct/nite/fd2log &

To restart *runacct*.

nohup runacct 0601 2>> /usr/adm/acct/nite/fd2log &

To restart *runacct* at a specific *state*.

nohup runacct 0601 MERGE 2>> /usr/adm/acct/nite/fd2log &

WARNINGS

Normally it is not a good idea to restart *runacct* in its **SETUP** *state*. Run **SETUP** manually, then restart via:

runacct *mmdd* **WTMPFIX**

If *runacct* failed in its **PROCESS** *state*, remove the last **ptacct** file because it will not be complete.

FILES

/usr/adm/acct/nite/active
/usr/src/cmd/acct/ctmp.h
/usr/adm/acct/nite/daytacct
/usr/adm/acct/nite/lastdate
/usr/adm/acct/nite/lock
/usr/adm/acct/nite/lock1
/usr/adm/pacct*
/usr/adm/acct/nite/ptacct*.*mmdd*
/usr/adm/acct/nite/statefile
/usr/src/cmd/acct/tacct.h
/etc/wtmp

SEE ALSO

mail(1), acct(1M), acctcms(1M), acctcom(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), acct(2), acct(4), utmp(4),

System Accounting in the *HP-UX System Administrator Manual.*

STANDARDS CONFORMANCE

runacct: SVID2

NAME

rusersd - network username server

SYNOPSIS

/usr/etc/rpc.rusersd [**-l** *log_file*] [**-e** | **-n**]

DESCRIPTION

rusersd is an RPC server that returns a list of users on the network. The **rusers** command prints this information (see *rusers*(1)).

inetd invokes **rusersd** through **/etc/inetd.conf** (see *inetd*(1M)).

Options

rusersd recognizes the following options and command-line arguments:

-l *log_file* Log any errors to the named log file, *log_file*. Errors are not logged if the **-l** option is not specified.

Information logged to the file includes date and time of the error, the host name, process ID and name of the function generating the error, and the error message. Note that different services can share a single log file since enough information is included to uniquely identify each error.

-e Exit after serving each RPC request. Using the **-e** option, the **inetd** security file **/usr/adm/inetd.sec** can control access to RPC services.

-n Exit only if

- **portmap** dies (see *portmap*(1M)),
- another **rpc.rusersd** registers with **portmap**, or
- **rpc.rusersd** becomes unregistered with **portmap**.

The **-n** option is more efficient because a new process is not launched for each RPC request. This option is the default.

AUTHOR

rusersd was developed by Sun Microsystems, Inc.

SEE ALSO

rusers(1), inetd(1M), portmap(1M), inetd.conf(4), inetd.sec(4), services(4).

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages.

NAME

rwall - write to all users over a network

SYNOPSIS

/usr/etc/rwall *hostname ...*
/usr/etc/rwall -n *netgroup ...*
/usr/etc/rwall -h *host* **-n** *netgroup*

DESCRIPTION

rwall reads a message from standard input until EOF, then sends the message, preceded by the line **Broadcast Message ...**, to all users logged in on the specified host machines. With the **-n** option, **rwall** sends the message to the specified network groups defined in **/etc/netgroup** (see *netgroup*(4)).

A machine can only receive such a message if it is running **rwalld**, which is normally started from **/etc/inetd.conf** by the **inetd** daemon (see *inetd*(1M)).

WARNINGS

The timeout is kept fairly short so that the message can be sent to a large group of machines (some of which may be down) in a reasonable amount of time. Thus, the message may not get through to a heavily loaded machine.

AUTHOR

rwall was developed by Sun Microsystems, Inc.

FILES

/etc/inetd.conf

SEE ALSO

rwalld(1M), shutdown(1M), wall(1M), netgroup(4).

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages.

NAME

rwalld - network rwall server

SYNOPSIS

/usr/etc/rpc.rwalld [**-l** *log_file*] [**-e**|**-n**]

DESCRIPTION

rwalld is an RPC server that handles **rwall** requests (see *rwall*(1)). **rwalld** calls **wall** to send a message to all users logged into the host on which **rwalld** is running (see *wall*(1)).

inetd invokes **rwalld** through **/etc/inetd.conf** (see *inetd*(1M)).

Options

rwalld recognizes the following options and command-line options:

-l *log_file* Log any errors to *log_file*. Errors are not logged if the **-l** option is not specified.

Information logged to the log file includes date and time of the error, the host name, process ID and name of the function generating the error, and the error message. Note that different services can share a single log file because enough information is included to uniquely identify each error.

-e Exit after serving each RPC request. Using the **-e** option, the **inetd** security file **/usr/adm/inetd.sec** can control access to RPC services.

-n Exit only if:

- **portmap** dies (see *portmap*(1M)),
- another **rpc.rwalld** registers with **portmap**, or
- **rpc.rwalld** becomes unregistered with **portmap**.

The **-n** option is more efficient because a new process is not launched for each RPC request. Note, this option is the default.

AUTHOR

rwalld was developed by Sun Microsystems, Inc.

SEE ALSO

inetd(1M), portmap(1M), rwall(1M), wall(1M), inetd.conf(4), inetd.sec(4), services(4).

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages.

NAME

rwhod - system status server

SYNOPSIS

`/etc/rwhod [-s] [-r]`

DESCRIPTION

rwhod is the server that maintains the database used by **rwho** and **ruptime** (see *rwho*(1) and *ruptime*(1)). **rwhod** sends status information to and receives status information from other nodes on the local network that are running **rwhod**.

rwhod is started at system boot time when the lines that invoke it are uncommented in the file **/etc/netbsdsrc**.

As an information sender, it periodically queries the state of the system and constructs status messages that are broadcast on a network.

As an information receiver, it listens for other **rwhod** servers' status messages, validates them, then records them in a collection of files located in the **/usr/spool/rwho** directory.

By default, **rwhod** both sends and receives information. **rwhod** also supports the following options:

-s Configures server to be an information sender only.

-r Configures server to be an information receiver only.

Status messages are generated approximately once every three minutes. **rwhod** transmits and receives messages at the port indicated in the **who** service specification (see *services*(4)). The messages sent and received, are of the form:

```
struct  outmp {
        char    out_line[8];            /* tty name */
        char    out_name[8];            /* user id */
        long    out_time;               /* time on */
};

struct  whod {
        char    wd_vers;
        char    wd_type;
        char    wd_fill[2];
        int     wd_sendtime;
        int     wd_recvtime;
        char    wd_hostname[32];
        int     wd_loadav[3];
        int     wd_boottime;
        struct  whoent {
                struct  outmp we_utmp;
                int     we_idle;
        } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order before transmission. System load averages are calculated from the number of jobs in the run queue over the last 1-, 5- and 15-minute intervals. The host name included is the one returned by the **gethostname()** system call (see *gethostname*(2)). The array at the end of the message contains information about the users logged in on the sending machine. This information includes the contents of the **utmp** entry for each non-idle terminal line and a value indicating the time since a character was last received on the terminal line (see *utmp*(4)).

rwhod discards received messages if they did **not** originate at a **rwho** server's port, or if the host's name, as specified in the message, contains any unprintable ASCII characters.

Valid messages received by **rwhod** are placed in files named **whod.***hostname* in the **/usr/spool/rwho** directory. These files contain only the most recent message in the format described above.

WARNINGS

rwhod does not relay status information between networks. Users often incorrectly interpret the server

dying as a machine going down.

AUTHOR

rwhod was developed by the University of California, Berkeley.

SEE ALSO

rwho(1), ruptime(1).

NAME

sa1, sa2, sadc - system activity report package

SYNOPSIS

/usr/lib/sa/sa1 [*t n*]

/usr/lib/sa/sa2 [**-ubdycwaqvmA**] [**-s** *time*] [**-e** *time*] [**-i** *sec*]

/usr/lib/sa/sadc [*t n*] [*ofile*]

DESCRIPTION

System activity data can be accessed at the special request of a user (see *sar*(1)) and automatically on a routine basis as described here. The operating system contains a number of counters that are incremented as various system actions occur. These include CPU utilization counters, buffer usage counters, disk and tape I/O activity counters, tty device activity counters, switching and system-call counters, file-access counters, queue activity counters, and counters for inter-process communications.

sadc and shell procedures *sa1* and *sa2* are used to sample, save, and process this data.

sadc, the data collector, samples system data *n* times every *t* seconds and writes in binary format to *ofile* or to standard output. If *t* and *n* are omitted, a special record is written. This facility is used at system boot time to mark the time at which the counters restart from zero. The **/etc/rc** entry:

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

writes the special record to the daily data file to mark the system restart.

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file **/usr/adm/sa/sa***dd* where *dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted. The following entries, if placed in **crontab** (see *cron*(1M)):

```
0 * * * 0,6  /usr/lib/sa/sa1
0 8-17 * * 1-5  /usr/lib/sa/sa1 1200 3
0 18-7 * * 1-5  /usr/lib/sa/sa1
```

produce records every 20 minutes during working hours and hourly otherwise.

The shell script *sa2*, a variant of *sar*, writes a daily report in file **/usr/adm/sa/sar***dd*. The options are explained in *sar*(1). The **crontab** entry:

```
5 18 * * 1-5  /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600 -A
```

reports important activities hourly during the working day.

The structure of the binary daily data file is:

```
struct sa {
        struct sysinfo si;      /* see /usr/include/sys/sysinfo.h */
        int  sztext;            /* current entries of text table  */
        int  szinode;           /* current entries of inode table  */
        int  szfile;            /* current entries of file table  */
        int  szproc;            /* current entries of proc table  */
        int  msztext;           /* size of text table  */
        int  mszinode;          /* size of inode table  */
        int  mszfile;           /* size of file table  */
        int  mszproc;           /* size of proc table  */
        long  textovf;          /* cumul. overflows of text table  */
        long  inodeovf;         /* cumul. overflows of inode table  */
        long  fileovf;          /* cumul. overflows of file table  */
        long  procovf;          /* cumul. overflows of proc table  */
        time_t  ts;             /* time stamp, seconds  */
        long  devio[NDEVS][4]; /* device info for up to NDEVS units  */
#define IO_OPS          0       /* cumul. I/O requests  */
#define IO_BCNT         1       /* cumul. blocks transferred */
#define IO_ACT          2       /* cumul. drive busy time in ticks  */
#define IO_RESP         3       /* cumul. I/O resp time in ticks  */
};
```

FILES

/tmp/sa.adrfl	address file
/usr/adm/sa/sa*dd*	daily data file
/usr/adm/sa/sar*dd*	daily report file

SEE ALSO

cron(1M), sar(1), timex(1).

STANDARDS CONFORMANCE

sa1: SVID2

sa2: SVID2

sadc: SVID2

NAME

sadp - disk access profiler

SYNOPSIS

sadp [**-th**] [**-d** *device* [*-drive*]] *s* [*n*]

DESCRIPTION

sadp reports disk access location and seek distance, in tabular or histogram form. It samples disk activity once every second during an interval of *s* seconds. This is done *n* times if *n* is specified. Cylinder usage and disk distance are recorded in units of 8 cylinders.

Options

sadp recognizes the following options and command-line arguments:

-t (default) Report data in tabular form.

-h Produce a histogram of the data.

-d *device* Profile a given device. Valid values for *device* are:

disc1 HP-IB disk,

disc2 Fiber Link (FL) disk,

disc3 Small Computer Systems Interface (SCSI) disk.

sadp can profile only one device type per invocation. The **-d** option can be omitted if the system has only one device type.

-drive

Specify the disk drive. *drive* can be:

- A driver logic unit number (lu) supported by *device*,
- Two lu numbers separated by a minus (indicating an inclusive range), or
- A list of driver lu numbers separated by commas.

If *drive* is not specified, **sadp** profiles all the disk drives specified by *device*. The lu number can be in the range of 0 through 255 for **disc1**, **disc2**, and **disc3** device drivers.

EXAMPLES

Generate four tabular reports, each describing cylinder usage and seek distance of **disc1** disk drive 0 during a 15-minute interval:

```
sadp -d disc1-0 900 4
```

FILES

/usr/lib/sadp	executable file on Series 800 systems
/dev/kmem	special file containing the image of kernel virtual memory

SEE ALSO

kmem(7), mem(7).

STANDARDS CONFORMANCE

sadp: SVID2

NAME

sam - system administration manager

SYNOPSIS

`/usr/bin/sam`

Remarks: At Release 9.0,

SAM has been tuned to run in the Motif environment (though it also runs in a text terminal environment). To run SAM using Motif windows:

1. Be sure Motif is loaded on the system on which SAM is being used (system must have `X11R5-PRG` and `X11R5-SHLIBS` filesets installed).
2. Set the `DISPLAY` environment variable to match the system name (e.g., `DISPLAY=`*system*`:0.0`).

DESCRIPTION

The `sam` command starts a menu-driven program that makes it easy to perform system administration tasks with only limited specialized knowledge of HP-UX. SAM is a self-guided tool, and context-sensitive help is available at any point by pressing the **f1** function key. Status messages show whether or not a task succeeded, and what errors occurred, if any.

To conveniently administer one or more remote systems from a local system, run SAM on the remote system with control from the local keyboard and display.

All actions by SAM are logged into the local system file `/usr/sam/log/samlog` or `/usr/sam/log/`*hostname*, depending on the environment. The [`Options`...] button in the SAM Control Box allows run-time control of logging.

SAM performs system administration tasks in the following areas:

User and Group Account Management:

- Add users, remove users, view/modify the account information, or change the password of any user on the system. You can "customize" the tasks of adding and removing users by specifying steps to be performed before and/or after SAM does its processing for the task. The `Task Customization` action items in SAM Users and Groups leads you through this capability. See "Customizing SAM Tasks" below for more information.
- Deactivate users (preventing them from logging in, but leaving their files and directories intact). These users can also be reactivated (allowing them to log in to your system again).
- Add groups, remove groups, and view or modify group membership.

Disk and File Systems Management:

- Add, move into the LVM (Logical Volume Manager), remove, or change address of hard disk drive.
- Add, modify, convert to long file names, or remove a local file system on a hard disk or on a logical volume.
- Remote (NFS) file systems configuration, including:
  - Add, modify, or remove remote (NFS) file systems.
  - Allow or disallow access by remote systems to local file systems.
  - Modify RPC (Remote Procedure Call) services' security.
- Add, remove, or modify a device swap on hard disk or logical volume.
- Add, mount, umount, or create a file system on an optical disk.
- Examine, create, extend, or reduce a volume group pool of disks.
- Create, extend or change number of mirrored copies of a logical volume and associated file sytem.

- Remove or reduce a logical volume.
- Split or merge mirrored copies of a logical volume.

Peripheral Devices Management:

- Add, modify, or remove the configuration of printers and plotters.
- Administer the LP Spooler.
- Add, modify, or remove the configuration of hard disk drives and CD-ROM drives.
- Add, remove, or modify device swap.
- Add file system swap.
- Add or remove terminals and modems.
- Add or remove tape drives.
- Add or remove hardware interface cards and HP-IB instruments (Series 800 only).
- View current configuration of peripherals and disk space information.

Backup and Recovery:

- Interactively back up files to a valid backup device (cartridge tape, cartridge tape autochanger, magnetic tape, DAT, magneto-optical disk or magneto-optical disk autochanger). The SAM interface is suspended so that you can read and/or respond to the interactive messages produced by **fbackup** (see *fbackup*(1M)).
- Recover files online from a valid backup device. The SAM interface is suspended so that you can read/respond to the interactive messages produced by **frecover** (see *frecover*(1M)).
- Add to, delete from, or view the automated backup schedule.
- Obtain a list of files from a backup tape.
- View various backup and recovery log files.

Process Management:

- Kill, stop or continue processes.
- Change the nice priority of processes.
- View the current status of processes.
- Schedule periodic tasks via cron.
- View current periodic (cron) tasks.
- Run performance monitors.

Routine Tasks: Shut down the system.

- View and remove large files. Specify size and time-since-accessed of large files to display or remove.
- View and remove core files.
- View and trim ASCII or non-ASCII log files. Add or remove files from the list of files to monitor. Set recommended size for trimming.

HP-UX Cluster Configuration (Series 300/700 only):

- Create an HP-UX cluster by setting up the cluster server (kernel, dfile, data files, and CDFs) and adding cluster clients.
- Add or remove cluster clients. You can customize the tasks of adding and removing cluster clients by specifying steps to be performed before and/or after SAM does its processing for the task. The **Configure Task Customization** action leads you through this capability. See "Customizing SAM Tasks" below for more information.

- Modify swap server system for cluster clients. Any client with a local swap disk can be configured as a swap server for other clients at the time the disk is added. Later, other clients can be set to swap to the swap serving client.

Kernel and Device Configuration:

- Change the configuration for I/O device drivers.
- Modify operating system parameters.
- Modify swap configuration in the kernel.
- Modify dump device configuration in the kernel.
- Add or remove optional subsystems such as NFS, LAN, NS, CD-ROM, etc.
- Generate a new kernel.
- Change your system console device (Series 700/800 only).

Networks/Communications:

- Configure the LAN card or cards and the services they support:
  - ARPA services.
  - Network Services (NS).
  - Network File System (NFS).
- Configure X.25 card or cards and PAD (Packet Assembler/Disassembler) services.
- Add, limit, or remove the ability of your system to communicate with remote systems via UUCP.

Auditing and Security (Trusted Systems):

- Turn the Auditing system on or off.
- Set the parameters for the Audit Logs and Size Monitor.
- View all or selected parts of the audit log(s).
- Modify (or view) which users, events, and/or system calls get audited.
- Convert your system to a Trusted System.

Networking Services and Trusted Systems features are not present on all system configurations, so these areas are not always accessible through SAM.

File System Protection When Removing Users

When removing users or files from a system, there is always the unfortunate likelihood that the wrong user may be removed or that files belonging to a user who is removed are deleted inadvertently during the removal process. For example, user **bin** is the owner of (from the operating system's perspective) the majority of the executable commands on the system. Removing this user would obviously be disastrous. On the other hand, suppose user **joe** owns all of the files comprising the test suite for a project. It may be appropriate to remove **joe**, but the test suite should be left intact and assigned to a new owner. SAM provides two features to help protect against inadvertent removal of users or files when removing users:

- When prompting for the name of a user to remove from the system, SAM checks the name given against a list of names specified in the file **/usr/sam/config/rmuser.excl**. If the name matches one within the file, SAM does not remove the user.
- When SAM removes a user, all files (or a subset thereof) for that user are also removed, unless the ownership is given to another user. Before removing a file belonging to the user, SAM checks to see if the file resides in a path that has been excluded from removal. SAM uses the file **/usr/sam/config/rmfiles.excl** to determine which paths have been excluded from removal. So, for example, if the path **/users/joe/test** is named in the file, SAM will not remove any files residing beneath that directory. SAM logs a list of all files it removes in the file **/tmp/sam_remove.log**.

- SAM does not remove or reassign any files if the user being removed has the same user ID (uid) as another user on the system.

Files **/usr/sam/config/rmuser.excl** and **/usr/sam/config/rmfiles.excl** can be edited to contain users and directories that you want to exclude from removal by SAM.

Customizing SAM Tasks

You can customize the following SAM tasks:

- Add a New User Account to the System
- Remove a User Account from the System
- Add a Cluster Client
- Remove a Cluster Client

For each of these tasks, you can specify steps you want performed before and/or after SAM does its processing for the task. Before SAM performs one of the tasks, it checks to see if a pre-task step (executable file) was defined. If so, SAM invokes the executable, passes it a set of parameters (see below), and waits for its completion. You can halt SAM's processing of a task by exiting from your executable with a non-zero value (for example if an error occurs during execution of your executable).

After SAM has finished processing, it checks for a post-task step, performing the same type of actions as for the pre-task step.

The executable file must have these characteristics:

- Must be owned by root.
- Must be executable only by root, and if writable, only by root.
- Must reside in a directory path where all the directories are writable only by owner.
- The full pathname of the executable file must be given in the SAM data entry form.

The same parameters are passed from SAM to your program for both the pre-task and post-task steps. Here are the parameters passed for each task:

- Add a New User Account to the System:

  -l *login_name*
  -v *user_id*
  -h *home_directory*
  -g *group*
  -s *shell*
  -p *password*
  -R *real_name*
  -L *office_location*
  -H *home_phone*
  -O *office_phone*

The file **/usr/sam/config/ct_adduser.ex** contains an example of how to process these parameters.

•

Remove a User Account From the System

There can be one of three possible parameters, depending on the option selected in the SAM data entry form. The parameter can be *one* of these three:

-f *user_name* option supplied when all of *user_name*'s files are being removed.

-h *user_name* option supplied when *user_name*'s home directory and files below it are being removed.

-n *new_owner user_name*
option supplied when all of *user_name*'s files are being assigned to

new_owner.

The file **/usr/sam/config/ct_rmuser.ex** contains an example of how to process these parameters.

•
Add a Cluster Client

When adding multiple clients, the customized task is invoked once for each client. If any pre-task command fails (returns non-zero), the corresponding client is not added.

The parameters are:

-h *client_nodename*	Name of the cluster client being added.
-m *machine_type*	**HP68020** for Series 300/400 or **HP-PA** for Series 700.
-i *internet_address*	Unique network address for the cluster client, in the form *ddd.ddd.ddd.*.
-a *link_level_address*	12-character hardware address associated with the LAN card in the cluster client.
-n *number_csps*	Number of cluster server processes allowed for the cluster client. Normally set to a default value of 1 for new clients.

The options **-c** *cluster_type*, **-l** *lan_card*, and **-f** *template_file* supported in previous releases are no longer supported.

The file **/usr/sam/config/ct_addclient.ex** contains an example of how to process these parameters. The task customize command is run with standard output and standard error sent to the cluster log file, **/tmp/cluster.log**.

•
Remove a Cluster Client

When removing multiple clients, the customized task is invoked once for each client. If any pre-task command fails (returns non-zero), the corresponding client is not removed. The format of the parameter string for this task is the same as for adding a cluster client, with the following additions:

-d *cnode_id*	Cnode ID of the client to remove.
-s *swap_id*	Cnode ID of the swap server system for the client to remove.
-r *remove_flag*	One of the values **REMOVE_FILES** or **REMOVE_KEEP** indicating whether the removed client's CDF elements will be removed or kept.

The options **-c** *cluster_type*, **-l** *lan_card*, **-f** *template_file*, and **-q** *remove_files_yes_no* supported in previous releases are no longer supported.

File **/usr/sam/config/ct_rmnode.ex** contains an example of how to process these parameters. The task customize command is run with standard output and standard error sent to the cluster log file, **/tmp/cluster.log**.

Adding New Functionality to SAM

You can easily add stand-alone programs and scripts to SAM. Under main menu item **Other Utilities**, you can create your own hierarchy of menus to access executable programs. SAM is suspended while the executable program is running. When it finishes, the SAM interface is restored. You can also write your own help screen for each menu item you create. To add functionality to SAM, create a text file with the following format and place it in directory **/usr/sam/custom** with a **.ou** extension on the file name (for example, **/usr/sam/custom/my_utils.ou**):

```
# Lines which start with a '#' are comments
label "menu name" {
   label "submenu name" {
      label "item name"
         command "executable program"
         help "optional help file for item"
      label "item name"
         command "executable program"
         help "optional help file for item"
   } help "optional help file for submenu"
   label "submenu name" {
      label "item name"
         command "executable program"
         help "optional help file for item"
      label "item name"
         command "executable program"
         help "optional help file for item"
   } help "optional help file for submenu"
} help "optional help file for menu name"
```

A very simple Other Utilities file would be:

```
label "Edit the message of the day"
   command "vi /etc/motd"
   help "/usr/sam/custom/somehelp.hlp"
```

An example of how to add menu items and help screens is located under the `Other Utilities` main menu item.

DEPENDENCIES

SAM runs in an X Windows environment as well as on the following kinds of terminals or terminal emulators:

- HP-compatible terminal with programmable function keys and on-screen display of function key labels.
- VT-100

Depending on what other applications are running concurrently with SAM, more swap space may be required. SAM requires the following amounts of internal memory:

8 Mbytes	if using terminal based version of SAM.
16 Mbytes	if using Motif Windows version of SAM.

For more detailed information about how to use SAM on a terminal, see the first chapter of the *System Administration Tasks* manual.

AUTHOR

`sam` was developed by HP.

FILES

`/usr/sam/WORKSPACE`	directory for working space, including lock files (if a SAM session dies, it may leave behind a spurious lock file).
`/usr/sam/bin`	directory containing executable files.
`/usr/sam/lib`	directory for internal configuration files.
`/usr/sam/log`	directory containing log files.
`/usr/sam/help/$LANG`	directory containing SAM language specific, online help files.
`/usr/sam/custom`	directory where users can put their `Other Utilities` files (`.ou` extensions) which will automatically be read by SAM at startup.
`/usr/sam/config`	directory containing configuration files.

`/usr/sam/config/rmuser.excl`
file containing a list of users that are excluded from removal by SAM.

`/usr/sam/config/rmfiles.excl`
file containing a list of files and directories that are excluded from removal by SAM.

SEE ALSO

Related manuals:

System Administration Tasks
Installing and Administering ARPA Services
Installing and Administering LAN/9000
Installing and Administering NFS Services
Installing and Administering Network Services
Installing and Administering X.25/9000
How HP-UX Works: Concepts for the System Administrator

NAME

savecore - save a core dump of the operating system

SYNOPSIS

Series 300, 400, and 700:

/etc/savecore [**-nvcpx**] [**-d** *dumpsystem*] [**-t** *tapedevice*] *dirname* [*system*]

Series 800:

/etc/savecore [**-nvcpxfiukS**] [**-w**[**0**|**1**|**2**]] [**-F** *corefile*] [**-d** *dumpsystem*] [**-t** *tapedevice*] *dirname* [*system*]

DESCRIPTION

savecore saves a core dump of the system (assuming one was made when the system crashed) and writes a reboot message in the shutdown log file. **savecore** should be executed toward the end of the **/etc/rc** file.

dirname is the name of the existing directory in which to store the core dump.

system is the name of a file containing the image of the current running system; that is, the system that is running when **savecore** is executed. If *system* is not specified, **/hp-ux** is assumed.

savecore checks the core dump to verify that it corresponds to *dumpsystem*. If it does, **savecore** saves the core image in the file *dirname* **/hp-core.***n* and a copy of *dumpsystem*, which contains the namelist, in the file *dirname* **/hp-ux.***n*. The trailing *n* in the path names is a number that increases by one every time **savecore** is run in that directory. This number is kept in the file *dirname* **/bounds**, which is created if it does not already exist.

Before **savecore** writes out a core image, it first checks the space available on the filesystem containing *dirname*. If there is not enough space available to save the complete core image and the **-p** option was not specified, **savecore** prints a message and quits. Space can be reserved in a filesystem by specifying the number of 512-byte blocks in a file named *dirname* **/minfree**. The **minfree** file is useful for ensuring enough file system space for normal system activities after a panic.

savecore also writes a reboot message in the shutdown log file, if one exists. (If a shutdown log file does not exist, **savecore** does not create one.) If the system crashes as a result of a panic, **savecore** also records the panic string in the shutdown log.

Options

-n No copy of the *dumpsystem* is saved in *dirname* **/hp-ux.***n*. If **-n** is used, the user must remember which kernel (for example, **/hp-ux**) corresponds to the saved core file. The core file alone is not very useful.

-v Additional messages are printed under some conditions. This option is usually used only for debugging.

-c Clear the dump device flag to indicate that the device no longer contains any useful dump information. The **-c** option is useful for manually inhibiting dump actions called by **/etc/rc**.

-p Execute a partial dump. The destination file accepts as much of the dump as disk space allows, then clears the dump device flag as though the entire dump succeeded. This is useful when disk space is very low, but some information is still desired.

-x This option is used to extract a core image and system from a specified mag tape device. A mag tape device must be specified to use this option. If **-t** is not specified, **savecore** prints an error message and exits.

-d *dumpsystem*

dumpsystem is the name of a file containing the image of the system that produced the core dump (that is, the system running when the crash occurred). If **-d** is not specified, **savecore** assumes *dumpsystem* is identical to *system*. This option is used when the system being booted to save the core dump differs from the system that crashed. This is usually necessary only when debugging new systems that are not stable enough to boot and run **savecore**.

-t tapedevice

This option is used to identify a mag tape device. If **-t** is specified without the **-x** option,

the core image and the system will be written to *tapedevice* . If this option is specified with the **-x** option, the core image and system will be read from the *tapedevice* and written to *dirname*. If both the **-n** and **-t** options are specified, **savecore** prints an error message and exits.

RETURN VALUE

Upon exit, **savecore** returns the following values:

0 A core dump was found and saved.
1 A core dump could not be saved due to an error or **minfree** limitation.
2 No core dump was found to save.

WARNINGS

Some implementations place the core dump in the disk swap area while the system reboots. On such systems, if too many programs are swapped out before **savecore** is run, **savecore** might be unable to recover the crash dump.

savecore cannot recover the crash dump if more than three days have elapsed since the crash occurred. In this case, **savecore** displays the message, **Dump time is unreasonable**.

When the **-d** option is specified, some implementations require that *system* and *dumpsystem* be configured similarly. For example, with some implementations, swap devices must be identically configured, and the amount of physical memory in the system must not change between the time of the crash and the running of **savecore**.

Prior to HP-UX Release 9.0, **savecore** could overflow available file system space while saving the core file. Beginning at Release 9.0, **savecore** first checks the available space and the file *dirname* **/minfree** (if it exists) before writing out the core file. If a *dirname* **/minfree** file was created prior to Release 9.0 to prevent **savecore** from consuming excessive file system space, be sure to reexamine the file to make sure it now contains only the number of 512-byte blocks of free space to be available after **savecore** finishes writing the core file.

DEPENDENCIES

Series 800:

Multiple dump devices can be configured on Series 800 systems so that large memory configurations can be dumped to a core file that is spread across more than one device if necessary.

By default, when the primary swap device is not used as one of the dump devices or when the core image on the primary swap device is saved, **savecore** runs in the background. This reduces system boot-up time by allowing the system to be run with only the primary swap device.

If the dump devices are also used as swap devices, **savecore** disables swapping to those devices by creating the file **/etc/savecore.LCK**. **swapon** does not enable the device for swapping if the device is locked in **/etc/savecore.LCK** (see *swapon*(1M) for more details).

As **savecore** finishes saving the image from each dump device, it updates the **/etc/savecore.LCK** file and executes **swapon** to enable swapping on the the device.

Series 800 Only Options:

On Series 800 systems, the following additional options and command-line arguments are recognized:

-f Run **savecore** in the foreground only. By default, **savecore** runs in the background when the primary swap device does not contain a portion of the core image. Turning this option on increases system boot-up time.

-i Selectively retrieve the core file if necessary. With this option, **savecore** saves the complete core image if there is enough space on the file system that contains *dirname*. If there is insufficient space to save the complete core file, **savecore** saves the kernel pages and possibly user pages, depending on the space available on the file system. A compact core file is created unless the **-S** option is specified. This option is recommended because it tries to save as much important information as possible after each system panic. Both **-u** and **-k** override this option.

-u Force **savecore** to save user and kernel pages if space allows. Overrides the **-i** option. A compact core file is created unless **-S** is specified.

-k Force to save only kernel pages if space allows. Overrides the **-i** option. A compact core file is created unless **-S** is specified.

-w[*n*] Communicate with **swapon**. *n* can be one of the following values:

- **0** Do not run **swapon** from **savecore**. Use this option only when **swapon** is not executed from the **/etc/rc** file.
- **1** (default) Call **swapon** each time savecore finishes saving the image from each dump device. This option provides the most efficient use of swap space.
- **2** Only call **swapon** when **savecore** finishes saving the image file from *all* dump devices. If this option is used, no additional swap space other than the primary swap space is available until the complete core dump image is saved. This option provides a second chance to retrieve the core image when **savecore** fails on first attempt.

-S
Create a sparsely populated core file instead of a compact one.

-F *corefile*
Expand the compact core file. The name of *corefile* must have a **.I** suffix. This option is useful only when preparing the compact core file for analysis. **-S** and **-d** must be used with this option. The expanded core file (without the **.I** suffix) is saved in directory *dirname*.

AUTHOR

savecore was developed by HP and the University of California, Berkeley.

FILES

/hp-ux	current system
/usr/adm/shutdownlog	shutdown log
*dirname***/bounds**	crash dump number
*dirname***/minfree**	minimum free blocks on file system

SEE ALSO

adb(1).

NAME

scancore - scan system core dump

SYNOPSIS

scancore *corefile* [*system*]

DESCRIPTION

scancore is used to print information about a crashed system.

In order to use **scancore**, first obtain an image of the memory of the system after it crashes. This can be done using **savecore** (see *savecore*(1M).

scancore reads the relevant system data structures from the core image file and indexing information from **/hp-ux** (or the specified system file) to determine the scancore information and the state of the processes at the point of the crash. Output consists of the system message buffer, crash information block for each processor in the system, and the process table information. For each process, process table information includes the process table entry values, state of the process, open file descriptors, and the stack trace of the process.

If the corefile is a compacted file (see *savecore*(1M)), scancore automatically expands the corefile while reading.

WARNINGS

Since scancore operates on kernel data structures, it is not possible to run an old version of **scancore** on a new release of the operating system or vice-versa. **scancore** tries to ensure that the *corefile*, *systemfile*, and **scancore** versions match, and if they do not match issues a warning. **scancore** is likely to behave improperly if operated on mismatching versions of *corefile*, *system*, and **scancore**.

AUTHOR

scancore was developed by HP.

SEE ALSO

savecore(1M).

NAME

scsictl - control a SCSI device

SYNOPSIS

`scsictl` [`-akq`] [`-m` *mode*[= *value*]] ... [`-c` *command*] ... *device*

DESCRIPTION

`scsictl` provides a mechanism for controlling a SCSI device. It can be used to query mode parameters, set configurable mode parameters, and/or to perform SCSI commands. The operations are performed in the same order as they appear on the command line.

Mode parameters that take only a binary value may be specified as either `on` or `off`, or as their numeric equivalents (1 or 0, respectively).

device specifies the character special file to use.

Options

The following options are recognized:

`-a` Read the status of all mode parameters available.

`-k` Cause `scsictl` to continue processing arguments even after an error is detected. The default behavior is to exit immediately when an error is detected.

Command line syntax is always verified for correctness, regardless of the `-k` option. Improper command line syntax causes `scsictl` to exit without performing any operations on the device.

`-q` Causes **scsictl** to suppress the labels that are normally printed when mode parameters are displayed. Mode parameter values are printed, blank separated, in the same order as they appear on the command line.

`-m` *mode* Read the status of the specified *mode* parameter. Available modes are:

`immediate_report`
For devices that support immediate reporting, this mode controls how the device responds to write requests. If immediate report is enabled (`on`) write requests may be acknowledged before the data is physically transferred to the media. Disabling immediate report forces the device to await completion of any write request before reporting its status.

`ir` Abbreviated form of (synonym for) `immediate_report`.

`-m` *mode=value*
Set the mode parameter *mode* to *value*. The available mode parameters are listed above.

`-c` *command*
Cause the device to perform the specified command. Available commands are:

`sync_cache` For devices that have an internal write cache, this command causes the device to flush its cache to the physical medium.

`erase` For magneto-optical devices that support write without erase, this command can be used to pre-erase the whole surface (section 2) to increase data throughput on subsequent write operations. This command maintains exclusive access to the surface during the pre-erasure.

Mode parameters and commands need only be specified up to a unique prefix. When abbreviating a mode parameter or command, at least the first three characters must be supplied.

DIAGNOSTICS

Diagnostic messages are generally self-explanatory. If no options are supplied, a usage message is generated listing all available mode parameters and commands.

WARNINGS

Not all devices support all mode parameters and commands listed above. Changing a mode parameter may have no effect on such a device.

Issuing a command that is not supported by a device may cause an error message to be generated.

DEPENDENCIES

Series 800

`scsictl` is not supported on sequential-access devices using the tape driver.

When the system is rebooted, the Series 800 system always resets the value of the immediate report mode parameter to `off`. If `ioctl()` or `scsictl` is used to change the setting of immediate reporting on a SCSI device, the new value becomes the default setting upon subsequent configuration (e.g., opens) of this device and retains its value across system or device powerfail recovery. However, on the next system reboot, the immediate-report mode parameter is again reset to `off`.

The `immediate_report` mode applies to the entire device; the section number of the *device* argument is ignored.

To aid recovery, immediate reporting is not used for writes of file system data structures that are maintained by the operating system, writes to regular files that the user has made synchronous with `O_SYNC` or `O_SYNCIO`, and writes to a hard disk (but not a magneto-optical device) through the character-device interface.

SEE ALSO

diskinfo(1M), open(2), fcntl(2).

NAME

sdfdf - report number of free SDF disk blocks

SYNOPSIS

sdfdf *device ...*

DESCRIPTION

sdfdf prints out the number of free blocks and free inodes available for SDF file systems by examining the counts kept in the super-blocks. *device* must be specified by device name.

AUTHOR

sdfdf was developed by HP.

SEE ALSO

du(1), df(1M), sdf(4).

NAME

sdffsck - SDF file system consistency check, interactive repair

SYNOPSIS

sdffsck [**-y**] [**-n**] [**-s**] [**-d**] *SDFdevice* ...

DESCRIPTION

sdffsck is intended to mimic the Series 500 implementation of *fsck*.

sdffsck checks and interactively repairs inconsistent conditions for SDF file systems. If the file system is consistent, then the number of files, the number of blocks used, the number of blocks free, and the percent of volume unused are reported. If the file system is inconsistent, the operator is prompted for concurrence before each correction is attempted. Note that many corrective actions will result in some loss of data. The amount and severity of the loss can be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission, *sdffsck* defaults to **-n**.

sdffsck makes multiple passes over the SDF file system, so care should be taken to ensure that the SDF device is quiescent.

Options

The following options are interpreted by *sdffsck*:

-y Assume a **yes** response to all questions asked.

-n Assume a **no** response to all questions asked; do not open the file system for writing.

-s Ignore the actual free list and unconditionally reconstruct a new one. This option is useful in correcting multiply claimed blocks when one of the claimants is the free list. When using this option, the number of unclaimed blocks reported by *sdffsck* includes all the blocks in the free map. This can produce extensive output if **-d** is also selected.

This option should only be selected after a previous *sdffsck* indicates a conflict between a file and the free map. After **sdffsck -s** has executed, the integrity of the conflicting file(s) should be checked.

-d Dump additional information. The more **d**s that are present, the more information that is dumped. Up to five **d**s can be specified. Using more than two, however, can result in an overwhelming amount of output.

sdffsck also recognizes, but ignores, the **-S** and **-t** options found in other versions of *fsck*. An appropriate warning is printed. The diagnostics are intended to be self-explanatory.

SDFdevice is a device file name describing the device on which the SDF file system to be checked resides (e.g., **/dev/rdsk/c1d1s4**).

Error messages from *sdffsck* are written to standard error. Information resulting from use of the **-d** option and normal output are both written to the standard output. Both standard output and standard error are unbuffered.

Inconsistencies checked include:

- Blocks claimed by more than one inode, or by the free list;
- Blocks claimed by an inode or the free list outside the range of the file system;
- Incorrect link counts;
- Blocks not accounted for anywhere;
- Bad inode format;
- Directory checks:

  Files pointing to unallocated inodes;
  Inode numbers out of range;
  Multiply linked directories;
  Link to the parent directory.

Orphaned files (allocated but unreferenced) with non-zero sizes are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory on the SDF file system. The name assigned is the inode number. The only restriction is that **lost+found** must exist in the root of the SDF file system being checked, and must have empty slots in which entries can be made. This is accomplished by executing

sdfmkdir SDFdev:/lost+found

(using the name of the SDF device for *SDFdev*).

Orphaned directories and files with zero size are, with the operator's concurrence, returned directly to the free list. This also happens if the **lost+found** directory does not exist.

WARNINGS

sdffsck cannot check devices with a logical block size greater than 4096.

AUTHOR

sdffsck was developed by HP.

SEE ALSO

sdfmkdir(1), sdf(4),

Series 500 HP-UX System Administrator Manual.

NAME

sdffsdb - examine/modify an SDF file system

SYNOPSIS

sdffsdb *SDFdevice*

DESCRIPTION

sdffdsb is intended to mimic the Series 500 implementation of **fsdb**.

sdffsdb provides a means for performing the following functions on the specified *SDFdevice*:

- Find the inode number of a file, given its full path name.
- Examine and modify the contents of the superblock (volume header).
- Examine and modify the contents of any inode or other file attribute.

Integer input to **sdffsdb** can be entered in decimal (default), octal (with a preceding **0**), or hexadecimal (with a preceding **0x**).

SDFdevice is a raw or block special file describing the device on which the SDF file system is located.

sdffsdb execution is interactive. Prompts consist of requests for the needed information. When execution begins, **sdffsdb** displays the following menu:

```
1 - find inode numbers.
2 - examine superblock.
3 - examine inodes.
q - quit.
```

after which you are requested to choose one of the options shown.

Typing **1** causes **sdffsdb** to accept full pathnames of files (relative to the door directory of the SDF file system); it returns the corresponding inode number. Typing **q** returns to the main menu.

Typing **2** displays the contents of each record in the superblock. Each record is numbered. If a right parenthesis **)** follows the number, the record can be modified. If a right curly bracket **}** follows the number, the record cannot be modified. You are then asked whether or not you want to modify the superblock. An answer beginning with **n** sends you back to the menu; an answer beginning with **y** causes **sdffsdb** to ask for the record number to be modified. If the record number specified cannot be modified, you are told about it, and prompted for another record number. If you specify a record number which can be changed, you are prompted for the new data. Typing **q** returns to the main menu.

Typing **3** causes **sdffsdb** to prompt you for a file attribute record number. Upon receipt of a valid number, the contents of that record are displayed, and you are prompted for the information you want to change. Parentheses and curly brackets have the same meanings as described above. Typing **q** returns to the main menu.

Typing **q** at the main menu level terminates the **sdffsdb** command.

WARNINGS

sdffsdb is deceptively easy to use, and therefore should be used with extreme care. Be sure you know what you are doing before you enter too deeply into options **2** or **3**. You are given the opportunity to abort any operation before you have changed anything (by typing **q**), so consider carefully what you are about to do before you do it. **sdffsdb** does not provide an "undo" function, and the changes you make are immediate.

sdffsdb cannot examine devices with a logical block size greater than 4096.

AUTHOR

sdffsdb was developed by the HP.

SEE ALSO

sdf(4).

NAME

sdsadmin - create and administer Software Disk Striping arrays

SYNOPSIS

Make an SDS Array Using Command-Line Options:

`sdsadmin` [`-f`] `-m` [`-T` *type*] [`-L` *label*] [`-P` *partition_size*] [`-S` *stripesize*] *disk1* [*disk2*] ...

Make an SDS Array Using a Configuration File:

`sdsadmin` [`-f`] `-m` [`-L` label] `-C` *description_file disk1* [*disk2*] ...

Import an SDS Array:

`sdsadmin -i` *disk1* [*disk2*] ...

Check an SDS Array:

`sdsadmin -c` *lead_device*

List Information about an SDS Array:

`sdsadmin` [`-l`] *device*

Destroy SDS Configuration Data on a Device:

`sdsadmin` [-f] `-d` *device1* [*device2*] ...

Undestroy SDS Configuration Data on a Device:

`sdsadmin -u` *device1* [*device2*] ...

DESCRIPTION

Use `sdsadmin` to:

- Create a Software Disk Striping (SDS) array,
- Import an existing array to a system that is configured differently,
- Perform miscellaneous utility functions on an existing array.

For a complete discussion of how to configure and administer SDS arrays, refer to the manual, *Improving Performance with Software Disk Striping*.

Software Disk Striping arranges data distribution on arrays of up to eight separate, identical disks so that they are functionally equivalent to one large disk. Each array can be divided into up to eight partitions.

SDS arrays can improve I/O performance because individual disks can process requests in parallel. SDS arrays also provide convenient management for disk partitions that are larger than a single disk, thus allowing very large file systems (up to 4 Gbytes (gigabytes)). Creating an SDS array on one disk does not improve performance, but does allow a single disk to have multiple partitions.

Each SDS array partition is striped symmetrically across disks in the array. **Striping**, instead of placing blocks 1 through *n* on a single disk, distributes data such that block 1 is on the first disk, block 2 is on the second disk, and so forth. **Stripesize** refers to the size of data blocks in each SDS partition, and is not related to the individual SCSI disk device internal (hardware) block size (typically 512 bytes).

The first disk in an SDS array is called the **lead device**, and is used as a handle to access partitions in an SDS array. See EXAMPLES below for information on how `sdsadmin` creates device special files to access partitions in an SDS array.

Options

`sdsadmin` recognizes the following options and command-line arguments:

disk Block or character device special file associated with a particular physical disk.

device Block or character device special file associated with any device in an SDS array (either a physical disk or any partition in the array).

`-f` Force an operation (used only with `-m` or `-d` option). This option overrides the disk device check and request for confirmation before proceeding when creating a new array or destroying existing array data on a device. Creating an SDS array on one or more disks that already belong to an array (disk contains SDS array data) or which contain an HP-UX file system is usually considered an error.

If this option is *not* used and the standard input device is a tty, `sdsadmin` asks for confirmation before creating an array on any disk containing an HP-UX file system; If standard input is not a tty, `sdsadmin` prints an error message and aborts without creating

the array.

Use the **-f** option with extreme caution. Creating an SDS array on a device that is already a member of another SDS array, or which contains an HP-UX file system effectively destroys all existing data on the device.

-m Create (make) a new SDS array on the specified disk or disks. An array consists of 1 to 8 identical disks (same manufacturer, model number, capacity, interface type, etc.), and the first disk specified becomes the **lead device** of the new SDS array.

To create SDS arrays, use command-line options (easier to execute) or using a configuration file (easier to determine at a later date what was done when the array was created). When using the command line, **sdsadmin** uses the specified *type*, *label*, *partition size*, and *stripesize* or their default values if not specified. When using a configuration file (**-C** option), configuration values are taken from the array description file or set to their default if they are not specified in the file. Using an array description file is more flexible because different partition sizes and stripe sizes can specified for each partition in the array. See *SDS Array Description Files* below for more information, including default values.

sdsadmin verifies that all disks to be used in the new SDS array are the same size, checks that the specified configuration parameters are valid, then writes data describing the new array configuration to each disk of the array.

For best performance, connect disks in the SDS array to two or more interfaces instead of using a single interface bus. **sdsadmin** then automatically interleaves the disks among available interfaces to create an optimal bus configuration. See the **-i** option for more information about optimal bus configurations.

When creating an SDS array, **sdsadmin** creates the necessary device files for each partition of the array in directories **/dev/dsk** (block special files) and **/dev/rdsk** (character special files). See EXAMPLES below for more information about device file creation. A message is printed to the standard output listing device files that can be used to access each partition in the new array.

sdsadmin also adds a new disktab entry for the SDS array to file **/etc/disktab** (see *disktab*(4)). The *type* field (see below) of the SDS array is used to name the disktab entry. If an entry of the same name already exists and is not identical to the new entry, **sdsadmin** prints an error message and aborts without creating a new array.

-T *type* Specify the *type* name for a new SDS array. This option is allowed only when using the **-m** option. See *SDS Array Description Files* below for a detailed description of the *type* parameter. When creating a new SDS array, specifying the *type* is optional. If no *type* is specified, **sdsadmin** assigns a *type* based on the characteristics of the SDS array. This option is not allowed when using the **-C** option.

-L *label* Specify the *label* for a new SDS array. This option is allowed only when using the **-m** option. See *SDS Array Description Files* below for a detailed description of the *label* parameter. When creating a new SDS array, a *label* is optional.

-P *partition_size*
Specify the partition size for a new SDS array. The total space in the SDS array is divided into partitions, each having the requested size. The last partition may be smaller than the requested size, depending on the total space available. Since an array can be divided into at most eight partitions, the partition size should be at least one-eighth of the total space in the SDS array. Otherwise, some space may remain unused. See *SDS Array Description Files* below for a detailed description of the *size* parameter, including the special keyword **max**. The default partition size is **max** (approximately 4 Gbytes or the capacity of the array, whichever is smaller). This option is not allowed when using the **-C** option.

-S *stripesize* Specify the stripe size for each partition of a new SDS array. This option is allowed only when using the **-m** option. For best results, *stripesize* should be the same as the file-system block size (not related to disk hardware block size). Default *stripesize* is 16 384 bytes. See *SDS Array Description Files* below for a detailed description of the *stripesize* parameter. This option is not allowed when using the **-C** option.

`-C` *description_file*

Specify the SDS array description file that describes the parameters of the new SDS array. This option is allowed only when using the `-f`, `-m`, and `-L` options. See *SDS Array Description Files* below for a detailed description of the SDS array description file. If an SDS array *label* is specified on the command line, it takes precedence over any `label` specification in the SDS array description file, thus making it possible to use the same description file to create multiple arrays while giving each array a unique name.

`-i`

Import an existing array. The configuration data stored on each disk in an SDS array includes the absolute device location (major and minor number) of all devices in the array. If an array is to be moved to different bus addresses than when it was originally created, the array must be *imported* to the new configuration.

To import an array, first connect all disks in the array to their new locations. When an SDS array is created, `sdsadmin` automatically orders the disks to create an optimal configuration. However, when importing an array `sdsadmin` cannot change the order of the disks, as this would corrupt the data contained on the array. Consequently, care should be used to ensure that the disks are connected in an optimal configuration. The performance of an SDS array is generally best when the disks of the array are interleaved rather than being connected to a single interface bus. For example, when installing a four-disk array on two busses, connect disk 1 to bus 1, disk 2 to bus 2, disk 3 to bus 1, and disk 4 to bus 2. `sdsadmin` issues a warning when disks are not connected in an optimal configuration. The array will still function, but performance may not be as good as if the disks were connected in an optimal configuration.

Once the devices in an SDS array are connected to the system, use `sdsadmin -i` to update the SDS array configuration information. Simply specify the device files corresponding to the new device locations on the command line (in any order, but see the previous paragraph for information about properly interleaving disks on two or more busses). The SDS configuration data on each disk of the array is updated to reflect the new device locations.

As with the `-m` option, `sdsadmin` creates the necessary device files to access the SDS array. An informational message is printed to the standard output that indicates what device files may be used to access each partition of the array. If necessary, `sdsadmin` also adds a new entry to `/etc/disktab` for the imported SDS array. If a `disktab` entry for the SDS array already exists but is not correct for the SDS array being imported, `sdsadmin` prints an error message and aborts the import operation.

`-c` *lead_device*

Check that the SDS array configuration information stored on *lead_device* is correct, and that all other disks in the array are present and accessible. *lead_device* is the block or character device special file associated with the **lead device** of an SDS array (either the physical disk or any partition of the array). If any errors are encountered, an appropriate error message is printed to the standard-error output. Warning messages are printed to the standard-error output if the device file specified is not the lead device of the SDS array, or if the devices are connected in a bus configuration that is not optimal. See description of the `-i` option for more information about optimal bus configurations.

`-l`

Print (to standard output) information about the SDS array. The device file given can specify any disk that is a member of the SDS array. The information includes:

- *label* (if any) of the array
- *type* of the array
- Which disk in the array the given disk is (1..8)
- The unique identifier assigned to this array (see below)
- Device addresses of all disks in the array
- Data on each partition of the array, including:
  - *stripesize* of the partition (if applicable)

- *size* of the partition

If the `-l` option is used to print information about the lead device of an SDS array, the data on the partitions will also include the names of the block and character device special files that are used to access each partition of the SDS array.

When an SDS array is created, **sdsadmin** assigns a unique 32-bit identifier to the array. This value is stored on each disk in the array, and provides verification that the disks are all members of the same array. Even when two arrays have the same **label** and **type**, **sdsadmin** can distinguish between the disks of the two arrays.

`-d`
Destroy the SDS configuration data stored on the given device(s). This option effectively destroys all data that is contained on the SDS array, and allows the disk to be used for other purposes. Only the SDS configuration data on the listed devices is destroyed, but *all data on the array is lost*. To destroy the SDS configuration data on all devices in an SDS array, use **sdsadmin -d** on all devices in the SDS array.

If the standard input device is a tty and the `-f` option has not been used, **sdsadmin** requests that the destroy operation be confirmed by printing a confirmation question to standard output then reading a response from the tty. If the response begins with a **y** the SDS data on the disk is destroyed. Any other response causes **sdsadmin** to abort the operation.

This option should be used with extreme caution. Once SDS configuration data on any disk in an SDS array has been destroyed, all data stored on the SDS array is lost.

`-u`
Undestroy the SDS configuration data stored on the given device(s). If the device has not been used to store other data since running **sdsadmin -d**, the SDS configuration data that was originally stored on the device is restored. If all disks of the array that were destroyed are successfully undestroyed, existing data stored on the array is again accessible. If the SDS configuration data cannot be restored, an error message is printed to the standard error output.

SDS Array Description Files

To create more complex SDS arrays than can be created simply by using the `-T`, `-P`, and `-S` options, **sdsadmin** requires an SDS array description file. The file is supplied to **sdsadmin** by using the `-C` option.

Before creating an SDS array description file, decide first how the SDS array should be configured. Factors to consider are: number of disks, number of partitions, and the size and stripe size of each partition. Once SDS array configuration is determined, the next step is to create an SDS array description file that **sdsadmin** uses to create the SDS array. Refer to the manual *Improving Performance with Software Disk Striping* for more information on determining array configuration and creating SDS array description files.

An SDS description file is an ordinary text file that can be created with any standard editor such as **vi** (see *vi*(1)). The file consists of (keyword, value) pairs that describe attributes of the SDS array. The file can also contain comments. A comment begins with the `#` character and continues to the end of the current line.

Description file keywords are defined as follows:

type The value of this field is used in the generation of disktab entries for the SDS array (appended to the prefix `HP_`). It is also used as the value for the **product id** field returned by *diskinfo*(1M). The *type* field may contain up to 16 characters from the set of letters (`A-Z a-z`), numbers (`0-9`), hyphen (`-`), period (`.`) and underscore (`_`).

type can be any value, subject to the above limitations and provided the value does not already exist in the file **/etc/disktab**. Choosing a *type* name that reflects how the SDS array is configured is often helpful.

If you do not specify a **type** for the SDS array, **sdsadmin** assigns a default **type** composed of the **product id** of the lead device followed by **x***number_of_disks*-*number_of_partitions*. For example, an SDS array of three HP97560 disks that has been divided into 2 partitions would have a default **type** value of **97560x3-2**.

label This optional field is used to label SDS arrays, and serves only as an identification mechanism. When used, the *label* field contains up to 16 characters from the set of letters (`A-Z a-z`), numbers (`0-9`), hyphen (`-`), period (`.`) and underscore (`_`).

label can be any value, subject to the above limitations. The *label* provides a convenient means for distinguishing between multiple SDS arrays. It is suggested that you always specify a **label** for SDS arrays, but this is not a requirement.

partition This keyword indicates the partition to which the subsequent **stripesize** and **size** keywords refer. The partition number can be any value from **1** to **8**. It is not necessary that contiguous values be used. For example, partitions 1 and 3 can be defined without defining partition 2.

stripesize This field defines the stripe size for the current partition. Valid stripe sizes are **4096**, **8192**, **16384**, **32768**, and **65536** bytes. The default *stripesize* for multiple disk SDS arrays is 16 384 bytes.

Specifying **stripesize** for a single-disk SDS array is not allowed because single-disk SDS arrays are not striped.

If the specified stripe size is larger than 16 384 bytes, and the partition is to contain an HP-UX file system, be sure to read the information in the *NOTES* section, below, regarding *newfs*(1M) and *disktab*(4) limitations.

See *Choosing the Right Stripe Size*, below for more information regarding the correct stripe size to use for a given application.

size This field indicates the size (in bytes) for the current partition. The special value **max** indicates 4 294 967 295 bytes (approximately 4 Gbytes), or all remaining space available in the SDS array, whichever is smaller. A **size** must be specified for each partition of an SDS array.

Partitions being used as swap space or as raw devices must not exceed 2 Gbytes (many applications cannot access devices larger than 2 Gbytes).

Partitions used as HP-UX file systems are limited to 4 Gbytes.

The exact size of partitions in an SDS array may or may not match the requested size. **sdsadmin** adjusts the partition size to meet implementation defined constraints. Partitions smaller than 4 Gbytes are rounded up to a value that is a multiple of the number of disks multiplied by 1 megabyte. For example, partition sizes on a 4-disk array are always a multiple of 4 Mbytes. 4-Gbyte partitions are trimmed such that they are really only 4 Gbytes – 3072 bytes.

Numeric values for the **stripesize** and **size** fields can be followed by the letters **K**, **M**, or **G** which respectively represent a multiplication factor of 1024 (Kbytes), 1 048 576 (Mbytes) or 1 073 741 824 (Gbytes), respectively.

Here is an example SDS array description file:

```
# Example SDS array description file.
# This array's type is 'BIGx4-3', indicating it has 4 large disks
# (HP 97560 containing 1.3 Gbytes each) divided into 3 partitions
#
# This array has been labeled 'example-1', to distinguish it
# from other arrays.
#
type        BIGx4-3
label       example-1

partition 1
    size          max
partition 2
    size         900M
    stripesize    32k
partition 3
    size          max
    stripesize     4k
```

The example SDS array has been given a **type** value of **BIGx4-3**. The choice of this name is arbitrary. Use any string here (subject to the limitations specified in the description of the **type** keyword). However, remember that the value of this field is used to name the disktab entry for the SDS array, so it must not be the same as any existing value in **/etc/disktab**. If **type** is not specified, **sdsadmin** chooses a default value based on the type of the lead device. Since the lead device is an HP97560, the default **type** that would be used if not explicitly specified would be **97560x4-3**.

The example SDS array has been labeled with the string **example-1**. The choice of this name is arbitrary, since it is used only to identify given SDS arrays in a multiple-array environment. If a given SDS array has a **label**, its value is reported by **sdsadmin** when the **-l** option is used.

This SDS array description file defines an SDS array that is divided into three partitions. Since the keyword **max** is used for the size of partition 1, the first partition is approximately 4096 Mbytes (4 Gbytes) in size. Since no *stripesize* was specified, the first partition is striped using the default stripe size of 16 384 bytes. The second partition is set at 900 Mbytes, and striped in 32-Kbyte chunks.

The keyword **max** is also used for the size of partition 3. However, since there are only about 172 Mbytes of space remaining in this SDS array, the third partition is 172 Mbytes in size (remember that the array consists of four HP97560 disks, each of which has a capacity of about 1.3 Gbytes). The third partition is striped in 4-Kbyte chunks.

Choosing the Right Stripe Size

The correct stripe size to use on each partition of an SDS array depends on many factors. If you will use the partition as an HP-UX file system, choose a stripe size identical to the file system primary block size. Generally, the larger the file system block size, the better the performance. However, using a large file system block size can consume more disk space, especially if the file system contains many small files.

When using a partition for swap space, use a stripe size of 16 384 bytes, the optimum stripe size for swap space.

When using the partition for other purposes, such as an application that accesses the device directly, the correct stripe size depends on how the application uses the device. Only with knowledge of the internals of the application or through experimentation can you determine the optimum stripe size.

EXAMPLES

This example shows configuring an SDS array by specifying the parameters on the command line. Assume a single cabinet contains three HP97560 disks (approximately 1.3 Gbytes each). The disks are connected to the SCSI interface in EISA slot 1 at addresses 6, 5 and 4. To treat these three disks as one larger and faster device, create an SDS array from them using the command:

```
sdsadmin -m /dev/dsk/c41d6s0 /dev/dsk/c41d5s0 /dev/dsk/c41d4s0
```

In this example, we have specified the minimum number of parameters necessary to create an array. The array **type** defaults to **97560x3-1** since the lead device is an HP97560 disk and three disks have been combined into 1 partition. The partition *size* and *stripesize* default to the values **max** and **16384** bytes, respectively. Since the **-L** option was not used, this SDS array is not labeled.

The new SDS array has one partition with a capacity of approximately 3 876 Mbytes. The data is striped across the three disks, using the default *stripesize* of 16 384 bytes.

If necessary, **sdsadmin** adds a disktab entry for this new SDS array to the file **/etc/disktab** and creates block and character device files to access the one partition of the SDS array. In this example, the names of the device files would be **/dev/dsk/c41d6s1** and **/dev/rdsk/c41d6s1**.

To use a larger *stripesize* and provide a label for the array, use the command:

```
sdsadmin -m -L example -S 32k /dev/dsk/c41d6s0 \
        /dev/dsk/c41d5s0 /dev/dsk/c41d4s0
```

The SDS array still has one large partition, but is labeled as **example**, and the data is striped using a *stripesize* of 32 Kbytes rather than the default *stripesize* of 16 Kbytes.

This last example shows configuring an SDS array with the **-C** *description file* option. Assume that four HP97560 disks are connected to two SCSI interfaces. Two are connected to the interface in EISA slot 1 (at addresses 6 and 5), and the other two are connected to the interface in EISA slot 2 (also at addresses 6 and 5). Using the example SDS description file from above (typed and saved in a file named **example**), to create an array use the command:

```
sdsadmin -m -C example /dev/dsk/c41d6s0 /dev/dsk/c41d5s0 \
        /dev/dsk/c42d6s0 /dev/dsk/c42d5s0
```

Note that the disks have been specified in an order that does not create an optimal bus configuration. However, **sdsadmin** corrects this by reordering the devices so that they are properly interleaved among the available busses.

Device File Creation

As part of initializing the disks to be used as an SDS array, **sdsadmin** creates device files in **/dev/dsk** (block-special files) and **/dev/rdsk** (character special files) for each partition in the array. In the previous example, the device file **/dev/dsk/c41d6s0** refers to the lead device of the array. Since the array has three partitions, **sdsadmin** creates the following device files:

/dev/dsk/c41d6s1	block special, partition 1
/dev/dsk/c41d6s2	block special, partition 2
/dev/dsk/c41d6s3	block special, partition 3
/dev/rdsk/c41d6s1	character special, partition 1
/dev/rdsk/c41d6s2	character special, partition 2
/dev/rdsk/c41d6s3	character special, partition 3

These device special files can now be used to access each partition of the SDS array as if each was a single device. To view these device special files at any time after creating the array, use **sdsadmin -l** on the lead device of the array.

NOTES

Since the maximum file-system block and fragment size that can be represented by *disktab*(4) is 16 384 bytes, to create a file system with larger block and fragment sizes the desired sizes must be specified using the **-b** and **-f** options of the **newfs** command (see *newfs*(1M)).

A small amount of space is reserved at the beginning of each disk of an SDS array to store the SDS configuration information. For single-disk SDS arrays, the reserved amount is 16 384 bytes. For SDS arrays of multiple disks, the reserved amount is about 0.1 percent of the capacity of a single disk in the SDS array.

WARNINGS

A hardware failure on a single disk in an SDS array results in loss of **all** data stored on all disks in the SDS array. As with any disk, *regular backups of data stored on an SDS array is strongly recommended.*

Use the **-f** option with extreme caution. Data loss results if you create an SDS array on a device that is already a member of another SDS array or if you create an SDS array on a device that contains an HP-UX file system.

Use the **-d** option with extreme caution. Removing the SDS configuration data from any device of an SDS array causes all data stored on that array to be lost.

Once an SDS array has been configured, it is not possible to change array configuration without losing the data that is stored on it. To reconfigure an array, the data from each partition must be backed up to another storage area, then restored after the SDS array has been reconfigured.

Partitions being used as swap space or as raw devices must not exceed 2 Gbytes (many applications cannot access devices larger than 2 Gbytes).

Partitions used as HP-UX file systems are limited to 4 Gbytes.

AUTHOR

sdsadmin was developed by HP.

FILES

/etc/disktab
/dev/dsk/*
/dev/rdsk/*

SEE ALSO

vi(1), diskinfo(1M), mknod(1M), newfs(1M), disktab(4).

Improving Performance with Software Disk Striping.

(Requires Optional ARPA Services Software)

NAME

sendmail - send mail over the internet

SYNOPSIS

/usr/lib/sendmail [*mode*] [*flags*] [*address*]
newaliases
mailq

DESCRIPTION

Some electronic mail programs ("user agents") provide front ends for users to create and read mail; other programs ("delivery agents" and "receiving agents") transport mail between hosts or perform final delivery of local mail. *sendmail* accepts messages from user agents or receiving agents and routes them to their destinations via the appropriate delivery agents (or "mailers"). In addition, *sendmail* itself implements a delivery agent and receiving agent for the SMTP protocol running over TCP/IP.

By default, *sendmail* reads its standard input up to an end-of-file (EOF) or a line containing only a period (.). It then routes copies of the message to all of the recipient addresses on its command line. It determines how to route a message from the syntax and contents of the recipient addresses, according to configuration information in a *sendmail* configuration file, by default **/usr/lib/sendmail.cf**.

newaliases, which is the same as **/usr/lib/sendmail -bi** (see below) builds *sendmail*'s alias database from a text file, by default **/usr/lib/aliases**. Local addresses (local user names) are looked up in the alias database and expanded as necessary, unless the user name is preceded by a backslash (\). When the aliases file contains multiple entries for a given alias, only the last entry is used. Normally, the sender is not included in any alias expansions; in other words, if "joe" sends to "group", and the expansion of "group" includes "joe", the letter is not delivered to "joe".

Each line of the alias text file must be of the form:

alias : *mailing list*

Mailing lists can be continued onto multiple lines; each continuation line must begin with white space. A *mailing list* is a comma-separated list of one or more of the following:

user_name	Local user names occurring in alias expansions will themselves be looked up in the alias database unless they are preceded by backslash ('\').
remote_address	The remote address syntax understood by *sendmail* is configured in the *sendmail* configuration file, and typically includes RFC 822 style user@domain and UUCP style host!user.
filename	This must be an absolute pathname. *sendmail* appends a message to the file only if the directory in which it resides is readable and searchable by all, and only if the file already exists, is not executable, and is writable by all.
| "*command line*"	*sendmail* pipes the message as standard input to the specified command. Note that the double quotes (") are necessary to protect blanks in the command line.
:include:*filename*	*sendmail* reads *filename* for a list of recipient addresses and forwards the message to each.

If a file **.forward** exists in a user's home directory and is owned by the user, *sendmail* redirects mail for that user to the list of addresses in the **.forward** file. The addresses in **.forward** and **:include:** files can be anything that can appear as an address on the right side of an alias in the alias database. Note that the alias database is examined before a recipient's **.forward** file. This means that the **.forward** file is examined only if the recipient does not appear as an alias, or if a local alias resolves to that recipient.

Mail that is temporarily undeliverable is saved in a mail queue. By default, the mail queue is stored in the directory **usr/spool/mqueue**.

mailq, which is the same as **/usr/lib/sendmail -bp**, lists the entries in the mail queue. *sendmail* retries failed deliveries from the queue when called with the **-q** flag (see below).

sendmail uses *syslog*(3C) to log a record of its activities. Normally *syslogd* is configured to log messages from the *mail* facility in the file **/usr/spool/mqueue/syslog**. See *syslogd*(1M). The amount of detail reported can be configured with *sendmail*'s **L** option (see below).

(Requires Optional ARPA Services Software)

MODES

sendmail operates in exactly one of the following modes:

-ba Run in ARPANET mode. This is the same as **-bm** mode except that all input lines must be terminated with CR-LF, all output lines are terminated with CR-LF, and the "Sender:" and "From:" message header fields are examined for the name of the sender.

-bd Run as the *sendmail* daemon; run in the background listening for connections on the SMTP port, forking SMTP servers as necessary. Only one *sendmail* daemon (**-bd** mode) may be run at one time. Only the super-user is permitted to start the *sendmail* daemon. Typically the *sendmail* daemon is also run with the **-q***interval* flag. See below.

-bi Initialize the alias database. Create a *dbm*(3x) database from the alias text file, by default **/usr/lib/aliases**. If this database does not exist, *sendmail* will not do aliasing. The contents of the alias database can be examined with *praliases*(1). Only the super-user is permitted to initialize the alias database.

-bk Kill the *sendmail* daemon. This will kill a daemon started with **-bd** either with or without the **-q***interval* flag. It will not kill a queue-processing daemon started only with **-q***interval*. Only the super-user is permitted to run **sendmail -bk**.

-bm Route mail from standard input to the recipient addresses following any modes or flags on the command line. This is the default mode.

-bp Print a listing of the mail queue.

-bs Use the SMTP protocol, as described in RFC 821, on standard input. As in **-ba** mode, all output lines are terminated with CR-LF.

-bt Run in address test mode. In this mode, *sendmail* reads lines of the form:

ruleset[,*ruleset*...] *address*

and rewrites *address* with the specified *rulesets*. Address test mode is used for debugging configuration files.

-bv Verify names only; do not try to collect or deliver a message. Verify mode is used to validate users or mailing lists.

-bz "Freeze" the configuration file, to improve performance. This writes *sendmail*'s data space to the file (by default) **/usr/lib/sendmail.fc**. If this file exists, future instances of *sendmail* will read in this pre-interpreted data; otherwise, *sendmail* must re-interpret the configuration file each time it executes. Only the super-user is permitted to run **sendmail -bz**. On a cluster, **/usr/lib/sendmail.fc** is a context dependent file. The script **/etc/freeze** should be run to freeze the configuration file on each cnode of the cluster (see *cdf*(4) and *freeze*(1m)).

FLAGS

sendmail also recognizes the following flags:

-C*file* Use an alternate configuration file. The frozen configuration file, if any, is ignored. If this flag is specified, *sendmail* will reset its real and effective uid to the real uid of the executing user.

-F*fullname* Set the full name of the sender.

-f*address* Set the "From" address, in other words, define the sender of the mail. This succeeds only if the address following **-f** corresponds to the real user executing *sendmail*, or if the real user is a "trusted user", defined in the configuration file.

-h*N* Set the hop count to *N*. If the hop count is greater than 30, the mail will be returned with the error message **too many hops (30 max)**. By default, *sendmail* determines the hop count of a message by counting **Received** header lines in the message.

-n Don't expand aliases.

-o*xvalue* Set option *x* to the specified *value*. The processing options are described below.

(Requires Optional ARPA Services Software)

-q[*interval*] If *interval* is omitted, process the mail queue once. Otherwise, process the queue at the specified *interval*. *interval* is given as a tagged number, where **s** means seconds, **m** means minutes, **h** means hours, **d** means days, and **w** means weeks. For example, **-q1h30m** and **-q90m** both set the queue processing interval to one hour thirty minutes. Only the super-user is permitted to run the mail queue.

-r*address* An alternate and obsolete form of the **-f** flag.

-t Read the message header to determine recipients. **To:**, **Cc:**, and **Bcc:** header lines are scanned for recipients to send to. The **Bcc:** line is deleted before transmission. Any addresses in the argument list are suppressed.

-v Run in verbose mode. Alias expansions, connection attempts, SMTP transactions, etc. are logged to standard output.

-Z[*file*] Use an alternate frozen configuration file. If *file* is omitted, sendmail uses **sendmail.fc** in the current working directory. If this flag is specified, *sendmail* resets its real and effective uid to the real uid of the executing user.

PROCESSING OPTIONS

A number of processing options normally set in the configuration file can also be set on the command line for a particular invocation of *sendmail*. Options set on the command line override values set in the configuration file. Options that are either "on" or "off" are turned on with **-o***option***true** and turned off with **-o***option***false**. For example, **/usr/lib/sendmail -oifalse** turns off the "**i**" option for this run only. Setting any command line options other than **d**, **e**, **i**, **L**, **m**, **o**, **r**, **s**, or **v**, or use of the **M** option to set a macro other than **r** or **s** cause *sendmail* to reset its real and effective uid to the real uid of the executing user.

A*file* Generate or use the alias database based on alternate alias text file *file*.

c On mailers that are considered "expensive" to connect to, don't initiate immediate connection; instead, queue the message for later delivery.

d*x* Set the delivery mode to *x*. Delivery modes are:

- **i** interactive (synchronous) delivery;
- **b** background (asynchronous) delivery;
- **q** queue only; i.e., expect the messages to be delivered next time the queue is run.

D

Automatically rebuild the alias database if it is out of date and if the database files are writable by all.

e*x*

Set error processing to mode *x*. Valid modes are:

- **m** mail back error messages;
- **w** *write*(2) an error message to the user's terminal if the sender is logged in, otherwise mail it back;
- **p** print error messages to standard output (default);
- **q** throw away error messages, only indicating failures by returning a non-zero exit status.

If the text of the message is not mailed back by modes **m** or **w** and if the sender is local to this machine, a copy of the message is appended to the file **dead.letter** (if it exists) in the sender's home directory.

F*mode*

The mode to use when creating temporary files.

f

Save UNIX-style "From" lines at the front of messages.

g*N*

The default group id to use when calling mailers.

H*file*

Use *file* as the SMTP help file.

(Requires Optional ARPA Services Software)

I
If set, if an MX record lookup or host name lookup fails because the nameserver is not running, *sendmail* defers the message. If not set, if an MX record lookup fails, *sendmail* assumes that the nameserver is not being used, and that there are no MX records, and try to connect directly to the host. If a host name lookup fails, *sendmail* assumes that there is no entry for the host and returns an error.

i
Do not interpret a period on a line by itself as a message terminator.

L*n*
Set log level to *n*. Log level 10 is usual.

M*xvalue*
Set the macro *x* to *value*. This option can only be used on the command line. The macro **w** (local host name) is a special case. The value is looked up using *gethostbyaddr* and **w** is defined as the canonical host name returned by *gethostbyaddr*. See *gethostent*(3N).

m
Include the sender in alias expansions.

n
If set, parse the right-hand sides of alias definitions when initializing the alias database. In this case, an error in a mailing list is reported when initializing the alias database. However, it may take a long time to initialize a large alias database. If not set, simply store the right-hand sides unexamined. In this case, an error in a mailing list is reported when an attempt is made to mail to it.

o
If set, assume that this message can have old-style headers (only spaces between addresses). If not set, assume that the message has new style headers (commas between addresses). If set, an adaptive algorithm is used that correctly determines the header format in most cases.

P*address*
Set the Postmaster address. If *address* is valid, when an undeliverable message is mailed back, a copy of the message header (but not the message body) is delivered to *address*.

Q*directory*
Queue messages in *directory*. The default is **/usr/spool/mqueue**.

q*N*
If the load average exceeds the limit set with the **x** option, *N* is divided by the difference (plus one) between the current load average and the **x** limit to determine the maximum priority value (i.e. minimum priority) of messages that will be delivered immediately. Messages with a higher priority value (i.e. lower priority) are queued . The default value of *N* is 10 000.

r*timeout*
The timeout on reads. If none is set, *sendmail* will wait forever for a mailer that is not responding. Refer to RFC 1123, "Requirements for Internet hosts Application and Support," section 5.3.2, for a discussion of SMTP timeouts.

S*file*
Specify the file in which to save statistics. The default is **/usr/lib/sendmail.st**. The saved statistics can be reported with *mailstats*(1).

s
Always instantiate the queue file, even under circumstances where it is not strictly necessary, such as when running in interactive delivery mode.

T*time*
Set the timeout on messages in the queue to the specified time. After sitting in the queue for this amount of time, messages are returned to the sender. The default is three days.

u*N*
Set real and effective uid to *N* when executing mailers. Set real (but not effective) uid to *N* when running in **-bd** mode.

v
Run in verbose mode.

x*N*
If the current five minute load average is greater than *N*, use the algorithm described under the **q** option to determine whether to queue messages rather than deliver them. The default value is 8.

X*N*
If the current five minute load average is greater than *N*, the *sendmail* daemon does not accept connections. The default value is 12.

y*N*
The value *N* is added to the priority of a message (thus *lowering* its priority) for each recipient. This penalizes messages with large numbers of recipients. The default value is 1000.

Y
If set, process each queued message in a separate process. If not set, *sendmail* keeps track of hosts that are down during queue processing, which may improve performance, but uses more memory.

z*N*
The result of multiplying the message precedence by the value *N* is subtracted from the message priority, thus favoring higher precedence messages. The default value is 1800.

Z*N*
Each time a message is processed from the queue, the value *N* is added to the priority of the message, thus penalizing queued messages that are undeliverable for long periods of time. The default value is 9000.

DIAGNOSTICS

sendmail outputs user error messages to standard output if the user attempts to execute sendmail with incorrect or inconsistent arguments or attempts to do something that is not allowed, or if mail delivery fails in error processing mode **p**. *sendmail* outputs system error messages both to standard output and to the *syslog mail* facility to report *sendmail* configuration errors, system call and other failures (such as inability to open files), failure of programs exec'ed by *sendmail*, etc. These messages are intended to be self-explanatory.

WARNINGS

sendmail normally runs setuid to root. This is disabled if the **-C** or **-Z** flag or certain options are specified on the command line. Therefore, if other than the super-user uses these options, or if *sendmail* is not running setuid, *sendmail* may not be able to write to the the queue directory (by default **/usr/spool/mqueue**) or the alias database files (by default **/usr/lib/aliases.dir** and **/usr/lib/aliases.pag**).

Note that in order for *sendmail* to route mail to filenames, command lines, and **:include:** specifications (via aliasing only), or to quoted user names (preceded by backslash), the sendmail configuration file must resolve such addresses to the local mailer.

AUTHOR

sendmail was developed by Eric Allman at the University of California, Berkeley.

FILES

Except for **/usr/spool/mqueue/syslog**, which is configured for *syslogd* in **/etc/syslogd.conf**, the following default filenames can be overridden by options in the configuration file or on the command line.

/usr/lib/aliases /usr/lib/aliases.pag /usr/lib/aliases.dir	alias text file alias data base files
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	SMTP help file
/usr/lib/sendmail.st	collected statistics
/usr/spool/mqueue	queue directory
/usr/spool/mqueue/syslog	log file
/usr/spool/mqueue/[dqtx]f*	mail queue

SEE ALSO

mail(1), mailx(1), elm(1), mailstats(1), praliases(1), freeze(1M), syslogd(1M), syslog(3C), dbm(3X), gethostent(3N), cdf(4).

NAME

setmnt - establish mount table /etc/mnttab

SYNOPSIS

`/etc/setmnt`

DESCRIPTION

`setmnt` creates the `/etc/mnttab` table (see *mnttab*(4)), which is needed for both the `mount` and `umount` commands (see *mount*(1M)). `setmnt` reads the standard input and creates an entry in `/etc/mnttab` for each line. Input lines have the format:

filesys node

where *filesys* is the name of the device special file associated with the file system (such as `/dev/dsk/c0d0s2`) and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the mount table entry.

WARNINGS

`mount` and `umount` rewrite the `mnttab` file whenever a file system is mounted or unmounted if `mnttab` is found to be out of date with the mounted file system table maintained internally by the HP-UX kernel. The `syncer` command also updates `mnttab` if it is out of date (see *syncer*(1M)).

`mnttab` should never be manually edited. Use of this command to write invalid information into `mnttab` is strongly discouraged.

`setmnt` silently enforces an upper limit on the maximum number of `/etc/mnttab` entries.

It is unwise to use `setmnt` to create false entries for `mount` and `umount`.

FILES

`/etc/mnttab`
table of mounted file systems

SEE ALSO

devnm(1M), mount(1M), mnttab(4).

STANDARDS CONFORMANCE

`setmnt`: SVID2

NAME

setprivgrp - set special attributes for group

SYNOPSIS

setprivgrp *group_name* [*privileges*]
setprivgrp -g [*privileges*]
setprivgrp -n [*privileges*]
setprivgrp -f *file*

DESCRIPTION

setprivgrp associates a group with a system capability, thus providing a means for providing access to certain super-user-like privileges to members of a particular group or groups. The command can take one of four forms as shown above.

Options

If no option is specified (first form), *group_name* is given access to the specified *privileges*.

The following options (remaining three forms) grant privileges to all groups or no groups, or obtain privilege information from a specified file:

-g *All* groups have access to the specified *privileges*.

-n *No* groups have access to the specifed *privileges*.

-f Privileges are granted as specified in the file identified by *file* which is usually **/etc/privgroup**.

Privileged Capabilities

System capabilities that can be granted to privileged groups by the **setprivgrp** command are:

RTPRIO Can use **rtprio()** for setting real-time priorities (see *rtprio*(2).

MLOCK Can use **plock()** for locking process text and data into memory, and the **shmctl() SHM_LOCK** function to lock shared memory segments (see *plock*(2) and *shmctl*(2).

CHOWN Can use **chown()** to change file ownerships (see *chown*(2).

LOCKRDONLY Can use **lockf()** to set locks on files that are open for reading only (see *lockf*(2).

SETRUGID Can use **setuid()** and **setgid()** to change, respectively, the real user ID or real group ID of a process (see *setuid*(2) and *setgid*(2).

If *privileges* is absent in the command line (or in *file* if the **-f** option is specified), any currently assigned privileges are removed for the corresponding group or groups. Note that capabilities set by this command are not added to existing capabilities for the same group. To add a capability for a particular group, you must respecify *all* capabilities that were already set for that group, as well as the new capability.

Group Privileges File Format

The file specifed with the **-f** option should contain one or more lines in the following format:

group_name [*privileges*]
-g [*privileges*]
-n [*privileges*]

ERRORS

setprivgrp returns 1 if the user is not super-user, and 2 if there is not enough table space to hold a new privileged group assignment.

WARNINGS

In the HP Clustered environment, group privileges apply only the the cluster node on which they are set. For example, to grant the **RTPRIO** privilege to members of group **project** In cluster clients **client1** and **client2**, run the **setprivgrp** command twice, once on **client1** and again on **client2**. The CHOWN privilege is different: if a group has **CHOWN** privilege on the cluster server, it will have the privilege on all cluster clients as well.

Only users with appropriate privileges can use the **setprivgrp** command.

AUTHOR

`setprivgrp` was developed by HP.

FILES

`/etc/privgroup`
`/etc/group`

SEE ALSO

getprivgrp(1), chown(2), getprivgrp(2), lockf(2), plock(2), rtprio(2), setuid(2), shmctl(2), privgrp(4), privilege(5).

NAME

showmount - show all remote mounts

SYNOPSIS

/usr/etc/showmount [**-a**][**-d**][**-e**][*host*]

DESCRIPTION

showmount lists all clients that have remotely mounted a filesystem from *host*. This information is maintained by the **mountd** server on *host* (see *mountd*(1M)). The default value for *host* is the value returned by **hostname** (see *hostname*(1)).

Options

-a Print all remote mounts in the format

name : *directory*

where *hostname* is the name of the client, and *directory* is the directory or root of the file system that was mounted.

-d

List directories that have been remotely mounted by clients.

-e

Print the list of exported file systems.

WARNINGS

If a client crashes, executing **showmount** on the server will show that the client still has a file system mounted. In other words, the client's entry is not removed from **/etc/rmtab** until the client reboots and executes:

```
umount -a
```

Also, if a client mounts the same remote directory twice, only one entry appears in **/etc/rmtab**. Doing a **umount** of one of these directories removes the single entry and **showmount** no longer indicates that the remote directory is mounted.

AUTHOR

showmount was developed by Sun Microsystems, Inc.

SEE ALSO

hostname(1), exportfs(1M), mountd(1M), exports(4), rmtab(4).

NAME

shutdown - terminate all processing

SYNOPSIS

/etc/shutdown [**-h** | **-r**] [**-d** *device*] [**-f** *lif_file*] [**-y**] [*grace*]

DESCRIPTION

shutdown is part of the HP-UX system operation procedures. Its primary function is to terminate all currently running processes in an orderly and cautious manner. **shutdown** can be used to put the system in single-user mode for administrative purposes such as backup or file system consistency checks (see *fsck*(1M)), and to halt or reboot the system. By default, **shutdown** is an interactive program.

Options

-h Shut down the system and halt.

-r Shutdown the system and reboot automatically.

-d *device* (Series 300 and 400 only) Reboot from the specified device. The device must be a LIF volume. The **-d** option can only be used with the **-r** option.

-f *lif_file* (Series 300 and 400 only) Reboot from the specified file. If *lif_file* is a pair of double quotes indicating an empty string (i.e., **-f ""**), a power-up search sequence is performed to find a system. Otherwise, *lif_file* must follow the LIF filename convention. The **-f** option can only be used with the **-r** option, and a space is required between the **-f** and its argument.

-y Do not require any interactive responses from the user. (Respond **yes** or **no** as appropriate to all questions, such that the user does not interact with the shut-down process.)

grace *grace* specifies, in seconds, a grace period for users to log off before the system shuts down. The default is 60 seconds. If *grace* is zero, **shutdown** runs more quickly, giving users very little time to log out.

If neither **-r** (reboot) or **-h** (halt) is specified, the system is placed in *run-level* **s**; see *init*(1M).

shutdown goes through the following steps:

- The **PATH** environment variable is reset to **/bin** and **/usr/bin**.
- The **IFS** environment variable is reset to space, tab, and newline.
- The user is checked for authorization to execute the **shutdown** command (only authorized users can execute the **shutdown** command; see FILES for more information on the **/etc/shutdown.allow** authorization file.)
- The current working directory is changed to the root directory (**/**).
- All file systems' super blocks are updated; see *sync*(1M). This must be done before rebooting the system to ensure file system integrity.
- The real user ID is set to that of the super-user.
- A broadcast message is sent to all users currently logged in on the system telling them to log out. The administrator can specify a message at this time; otherwise, a standard warning message is displayed.
- User-supplied custom scripts in the **/etc/shutdown.d** directory are executed. See FILES for more information on this directory.
- The accounting subsystem is shut down.
- The next step depends on which options are selected:
  - If the system is being **halted** or **rebooted** and it is not a *mirrored disc system* or an *auxiliary swap server* with a *local mounted file system*, **/etc/reboot** is executed to finish bringing down the system.
  - If the system is being brought down to *single-user* state or is a *mirrored disc system* or is an *auxiliary swap server* with a *local mounted file system*, the following steps occur:

- All currently executing processes are terminated (see *killall*(1M)).
- The auditing subsystem is turned off.
- All locally mounted file systems are unmounted.

The system is rebooted or halted by executing `/etc/reboot` if the `-h` or `-r` option was chosen. If the system was being brought down to single-user state, a signal is sent to the `init` process to change states (see *init*(1M)).

In the HP Clustered environment, executing `shutdown` on the cluster server of a cluster causes all cluster nodes to also shut down. Executing `shutdown` on an auxiliary swap server causes all swap clients to be shut down as well as the swap server.

Each cluster node is shut down by executing `/etc/reboot` locally.

If neither `-h` nor `-r` is specified from the root or auxiliary swap server, the client machines are still rebooted. Note that prior to executing `/etc/reboot` on each cluster node, the remote boot daemon on the root or auxiliary swap server is disabled. Thus, client cluster nodes cannot actually reboot until the remote boot daemon (`rbootd`) is enabled on the root or auxiliary swap server.

Executing `shutdown` on a client node which is not also an auxiliary swap server only affects that node.

DIAGNOSTICS

`device busy`
: This is the most commonly encountered error diagnostic, and happens when a particular file system could not be unmounted; see *mount*(1M).

`user not allowed to shut down this system`
: User is not authorized to shut down the system. User and system must both be included in the authorization file `/etc/shutdown.allow`.

EXAMPLES

Immediately reboot the system and run HP-UX again:

```
shutdown -r 0
```

Halt the system in 5 minutes (300 seconds) with no interactive questions and answers:

```
shutdown -h -y 300
```

Go to init *run-level* `s` in 10 minutes:

```
shutdown 600
```

FILES

`/etc/shutdown.allow`
: Authorization file for `/etc/shutdown`. Lines consist of a system hostname and the login name of a user authorized to reboot or halt the system. The super-user's login name must be included in this file in order to execute `shutdown`. However, if the file is missing or of zero length, the `root` user can run the `shutdown` program to bring the system down. This file does not affect authorization to bring the system down to single-user state for maintenance purposes; that operation is permitted only when invoked by the super-user.

  A comment character, `#`, at the beginning of the line causes the rest of the line to be ignored (comments cannot span multiple lines without additional comment characters). Blank lines are also ignored.

  A subset of the wildcards available in `hosts.equiv` are available in the `shutdown.allow` file (see *hosts.equiv*(4)). Specifically the `%` (all nodes in the cluster) and `+` (all hosts or all users) wildcards are available. The `%` wildcard is not valid for standalone systems. Example entries:

```
# user1 can shut down systemA and systemB
systemA user1
systemB  user1
# root can shut down the whole cluster
%  root
```

```
# Anybody can shut down systemC
systemC  +
```

/etc/shutdown.d
Directory for user-supplied scripts that are needed to gracefully shut down the system. Files in this directory are executed in machine-sequence (ASCII) order. Files whose names begin with a dot, subdirectories, and files in subdirectories are ignored by the shutdown process.

One way to control the order of execution of these files is to prefix numbers onto the file names. For example:

```
0010StopAppA
0030StopAppD
0100StopAppB
```

These names cause **0010StopAppA** to be executed first, followed by **0030StopAppD**, then **0100StopAppB**.

Files are executed using the Bourne shell. **shutdown** sets the environment for executing scripts as follows:

- **PATH** variable is set to **/bin** and **/usr/bin**,
- **IFS** variable is set to **space**, **tab**, and **newline**,
- Current working directory is the root directory,
- Real user ID is that of the super-user.

Permissions on the **/etc/shutdown.d** directory should not be changed, and careful control of this directory should be maintained in order to ensure system integrity.

DEPENDENCIES

Series 700 and 800
The **-d** and **-f** options are not supported.

WARNINGS

The user name compared with the entry in the **shutdown.allow** file is obtained using **getlogin()** or, if that fails, using **getpwuid()** (see *getlogin*(3) and *getpwuid*(3)).

The hostname in **/etc/shutdown.allow** is compared with the hostname obtained using **gethostbyname()** (see *gethostbyname*(3)).

shutdown must be executed from a directory on the root volume, such as the **/** directory.

The maximum broadcast message that can be sent is approximately 970 characters.

The maximum full pathname to scripts in the **/etc/shutdown.d** directory is 255 characters. Given the directory name, this means that file names in the directory must be shorter than 239 characters.

SEE ALSO
fsck(1M), init(1M), killall(1M), mount(1M), rbootd(1M), reboot(1M), sync(1M), gethostbyname(3), getpwuid(3), hosts.equiv(4).

NAME

sig_named - send signals to the domain name server

SYNOPSIS

sig_named [**-v**] [**debug** [**+**] *debug-level* | **dump** | **kill** | **restart** | **stats**]

DESCRIPTION

sig_named sends the appropriate signal to the domain name server **/etc/named**. The process ID is obtained from **/etc/named.pid** or from *ps*(1). if **/etc/named.pid** does not exist.

Options

sig_named recognizes the following options and command-line arguments:

-v Verify that the name server is running before sending the signal. The verification is done using **ps** (see *ps*(1)).

debug [**+**]*debug-level*
Set the debugging output sent to **/usr/tmp/named.run** to *debug-level*. If debugging is already on, it is turned off before the debug level is set. If **+** precedes *debug-level*, the current debugging level is raised by the amount indicated. If *debug-level* is zero, debugging is turned off.

dump Signal the name server to dump its database. The database is dumped to **/usr/tmp/named_dump.db**.

kill Kill the name server process.

restart Signal the name server to reload its database.

stats Remove the old statistics file, **/usr/tmp/named.stats**. Signal the name server to dump its statistics. Show the statistics file on the standard output.

AUTHOR

sig_named was developed by HP.

FILES

/etc/named.pid	process ID
/usr/tmp/named.run	debug output
/usr/tmp/named_dump.db	dump of the name server database
/usr/tmp/named.stats	nameserver statistics data

SEE ALSO

kill(1), named(1M).

NAME

snmpd - daemon that responds to SNMP requests

SYNOPSIS

snmpd [**-C** *contact*] [**-L** *location*] [**-a**] [**-c**] [**-k**] [**-l** *logfile*] [**-m** *logmask*]

snmpd [**-M** *logmask*]

DESCRIPTION

snmpd is the Simple Network Management Protocol (SNMP) agent.

An SNMP manager sends requests for Management Information Base (MIB) values to **snmpd** using the SNMP. The agent replies with the information requested.

The MIB is a conceptual database of values on the agent. See **/usr/OV/snmp_mibs/hp-unix** for a list of MIB values that **snmpd** supports. The MIB file **/usr/OV/snmp_mibs/hp-unix** is included with HP OpenView products. The agent supports all objects in RFC 1213 except the EGP group.

Only the super-user can start **snmpd**, and only one **snmpd** can execute at a time.

When invoked, **snmpd** reads **/etc/snmpd.conf** to configure itself (see *snmpd.conf*(4)).

Options

snmpd recognizes the following options:

-C *contact* Specify the contact person responsible for the network management agent. This option overrides the contact person specified in **/etc/snmpd.conf**.

-L *location* Specify the location of the agent. This option overrides the location specified in **/etc/snmpd.conf**.

-M *logmask* Send a message to the currently running **snmpd** to change its logging mask to *logmask*. See the Log Masks section for valid values.

-a Suppress sending authenticationFailure traps.

-c Reconfigure **snmpd** (force **snmpd** to re-read **/etc/snmpd.conf**).

-k Kill the currently running **snmpd**.

-l *logfile* Use *logfile* for logging rather than the default logfile, **/usr/adm/snmpd.log**.

-m *logmask* Sets the initial logging mask to *logmask*. See the Log Masks section for valid values.

Traps

The agent also sends information to a manager without an explicit request from the manager. Such an operation is called a "trap". **snmpd** sends the following SNMP traps:

coldStart Sends a coldStart trap when **snmpd** is invoked.

linkDown Sends a linkDown trap when an interface goes down.

linkUp Sends a linkUp trap when an interface comes up.

authenticationFailure
Sends an authenticationFailure trap when an SNMP request is sent to **snmpd** with a community name that does not match the community names specified in **/etc/snmpd.conf**.

Each SNMP request is accompanied by a community name, which is a password that enables SNMP access to MIB values on an agent. A manager can request to read a MIB value by issuing an SNMP GetRequest, or a manager may request to alter a MIB value by issuing an SNMP SetRequest.

Log Masks

Log masks specify the type of output listed in **/usr/adm/snmpd.log** or in *logfile*. To select multiple output types, add the individual *logmask* values together and enter that number.

0 Turn off logging.

1 Log authenticationFailure traps.

2 Log errors.

4 Log configuration requests.

8 Log requests and replies.

16 Log requests and replies for objects that have been added.

32 Log hexdumps of packets received and sent by **snmpd**.

64 Log trace messages.

Defaults

By default, the agent (**/etc/snmpd**) does not allow managers to alter MIB values (it returns errors for SNMP SetRequests). To configure the agent to respond to SNMP SetRequests, add a **set-community-name** to **/etc/snmpd.conf**.

By default, the agent responds to all SNMP GetRequests, regardless of community name used in the request. To configure the agent to respond to a specific community name, add a **get-community-name** to **/etc/snmpd.conf**.

By default, SNMP traps are not sent to any destination. To configure the agent to send traps to one or more specific destinations, add the trap destinations to **/etc/snmpd.conf**.

By default, the agent's location is a blank string. To configure the agent's location, add the location to **/etc/snmpd.conf**, or use the **-L** option.

By default, the agent's contact is a blank string. To configure the agent's contact, add the contact to **/etc/snmpd.conf**, or use the **-C** option.

By default the agent logs authenticationFailure traps and errors to **/usr/adm/snmpd.log** or in *logfile*.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages appear. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **snmpd** behaves as if all internationalization variables are set to "C." See *environ*(5).

International Code Set Support

Supports single-byte character code sets.

AUTHOR

snmpd was developed by HP and Massachusetts Institute of Technology.

FILES

/etc/snmpd.conf
/usr/adm/snmpd.log
/usr/OV/snmp_mibs/hp-unix

SEE ALSO

chksnmpd(1), snmpd.conf(4).

RFC 1155, RFC 1157, RFC 1212, RFC 1213.

NAME

spray - spray packets

SYNOPSIS

/usr/etc/spray *host* [**-c** *count*] [**-l** *length*]

DESCRIPTION

spray sends a one-way stream of packets to *host* using RPC, then reports how many were received by *host* and what the transfer rate was. The host name can be either a name or an internet address.

Options

spray recognizes the following options and command-line arguments:

-c *count* Specifies how many packets to send. The default value of *count* is the number of packets required to make the total stream size 100 000 bytes.

-l *length* The number of bytes in the Ethernet packet holding the RPC call message. Since the data is encoded using XDR, and XDR only deals with 32-bit quantities, not all values of *length* are possible. The *spray* command rounds up to the nearest possible value. When *length* is greater than the size of an Ethernet packet, the system breaks the datagram into multiple Ethernet packets. The default value of *length* is 86 bytes (the size of the RPC and UDP headers).

AUTHOR

spray was developed by Sun Microsystems, Inc.

SEE ALSO

ping(1M), sprayd(1M).

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages.

NAME

sprayd - spray server

SYNOPSIS

/usr/etc/rpc.sprayd [**-l** *log_file*] [**-e**|**-n**]

DESCRIPTION

sprayd is an RPC server that records the packets sent by **spray** from another system (see *spray*(1M)).

inetd invokes **sprayd** through **/etc/inetd.conf** (see *inetd*(1M)).

Options

sprayd recognizes the following options and command-line arguments:

-l *log_file* Log any errors to the named log file, *log_file*. Errors are not logged if the **-l** option is not specified.

Information logged to the file includes date and time of the error, host name, process id and name of the function generating the error, and the error message. Note that different services can share a single log file since enough information is included to uniquely identify each error.

-e Exit after serving each RPC request. Using the **-e** option, the **inetd** security file **/usr/adm/inetd.sec** can control access to RPC services.

-n Exit only if

- **portmap** dies (see *portmap*(1M)),
- Another **rpc.sprayd** registers with **portmap**, or
- **rpc.sprayd** becomes unregistered with *portmap*.

The **-n** option is more efficient because a new process is not launched for each RPC request. **-n** is the default.

AUTHOR

sprayd was developed by Sun Microsystems, Inc.

SEE ALSO

inetd(1M), spray(1M), portmap(1M), inetd.conf(4), inetd.sec(4), services(4).

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages.

NAME

statd - network status monitor

SYNOPSIS

`/usr/etc/rpc.statd` [`-l` *log_file*]

DESCRIPTION

`statd` is an RPC server. It interacts with `lockd` to provide crash and recovery functions for the locking services on NFS (see *lockd*(1M)).

Options

`statd` recognizes the following options and command-line arguments:

`-l` *log_file* Log any errors to the named log file, *log_file*. Errors are not logged if the `-l` option is not specified.

Information logged to the file includes date and time of the error, host name, process id and name of the function generating the error, and the error message.

FILES

`/etc/sm/*`
`/etc/sm.bak/*`
`/etc/state`

AUTHOR

`statd` was developed by Sun Microsystems, Inc.

SEE ALSO

fcntl(2), lockf(2), signal(2), lockd(1M), sm(4).

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages

WARNINGS

Changes in status of a site are detected only upon startup of a new status monitor and lock daemon.

NAME

stcode - translate hexadecimal status code value to textual message

SYNOPSIS

`/etc/ncs/stcode` *hex_stat_code*

DESCRIPTION

`stcode` prints the textual message associated with a hexadecimal status code. This command is useful when a program produces a hexadecimal status code instead of a textual message.

`stcode` processes predefined status codes. No provision is currently made to add user-defined status codes to the error text database.

hex_stat_code is the hexadecimal status code to be translated.

EXAMPLES

Translate the hexadecimal status code 1c010003:

```
$ stcode 1c010003
unknown interface (network computing system/RPC runtime)
```

NAME

subnetconfig - configure subnet behavior

SYOPNSIS

```
subnetconfig local
subnetconfig remote
subnetconfig
```

DESCRIPTION

subnetconfig is used to set the value of an internal flag which in turn controls subnet behavior on a given host. Subnetting is an addressing scheme for partitioning hosts in a particular network into different sub-networks. Depending on the setting of the internal flag, each subnet in a network is viewed as being a distinct network or as belonging to the same network when choosing maximum size of outbound TCP packets. The flag also affects some of the error codes returned when trying to reach unreachable hosts. **subnetconfig** has no effect on routing decisions.

TCP selects a maximum outbound packet size at connection setup time. When connecting to hosts in the same network, the maximum packet size is limited only by the MTU size of the interface on which packets will be sent. For connections to destinations not in the same network, TCP limits segments to 512 bytes. By default, all subnets of the same network are treated as being part of the same network.

Command Forms

The **subnetconfig** command can be used in any of the following three forms:

subnetconfig local
(default behavior)
Set the internal flag so that all destinations in the same network (destination network number equals host network number), including all subnets defined in the network, are treated as being in the local network when choosing TCP segment sizes. Outbound TCP packet size for destinations in the local network is determined by the capacity of the interface on which packets are sent for that connection. TCP packets sent to a destinations in a different network are limited to 512 bytes.

Returns error code EHOSTUNREACHABLE if attempt is made to send packets to an unreachable host in the same network.

subnetconfig remote
Set the internal flag so that all destinations in the same network (destinations network number equals host network number), that are not part of the directly attached subnet (destination subnet number not equal to host subnet number) are treated as being in a distinct network. TCP then selects a maximum packet size of 512 bytes for all connections to destinations that are not part of the directly attached subnet. The size of packets sent to the directly attached subnet remains unchanged (governed by the MTU of the interface).

Returns error code ENETUNREACHABLE if attempt is made to send packets to an unreachable host on a different subnet of the same network. When trying to send a packet to an unreachable host in the same subnet, EHOSTUNREACHABLE is returned.

subnetconfig Display the current setting of the internal flag.

AUTHOR

subnetconfig was developed by HP.

NAME

swapinfo - system swap space information

SYNOPSIS

/etc/swapinfo [**-mtadfhqw**]

DESCRIPTION

swapinfo prints information about device and file system swap space. By default, it prints to standard output a two line header as shown here, followed by one line per swap area:

```
             Kb      Kb      Kb      PCT     START/  Kb
TYPE         AVAIL   USED    FREE    USED    LIMIT   RESERVE  PRI     NAME
```

The fields are:

TYPE One of:

dev Swap space residing on a mass storage device, either outside the file system or consuming some or all of a raw device. This swap space is either statically declared in the kernel, or dynamically added using the **swapon** command (see *swapon*(1M)), often in **/etc/rc** during system initialization based on the contents of **/etc/checklist**.

fs Dynamic swap space available from a file system, as set by **swapon**, often as listed in **/etc/checklist**.

hold Swap space on hold. This is space not allocated on any specific device or file system, but nonetheless held by the kernel for future use "on demand" by existing processes that have not yet consumed all the swap space they might need. It cannot be reallocated until those processes terminate.

Kb AVAIL

The total available swap space from the device or file system, in blocks of 1024 bytes (rounded to nearest whole block if necessary), including any swap space already in use. For file system swap areas the value is not necessarily constant. It is the current blocks used for swapping, plus the free blocks available to ordinary users minus RESERVE (but never less than zero). AVAIL is never more than LIMIT if LIMIT is non-zero.

Kb USED

The current number of 1-Kbyte blocks used for swapping from the device or file system. Device swap areas are consumed in machine-dependent "chunks" larger than one block (based on the configurable kernel value *swchunk*; the chunk size defaults to 4 Mbytes on Series 700/800 systems and to 2 Mbytes on Series 300/400 systems). If Kb AVAIL is not a multiple of the number of blocks in a chunk, the left over fraction of a chunk is always shown as used, except on unused swap devices (see below).

Kb FREE

The difference between Kb AVAIL and Kb USED.

PCT USED

The percentage of capacity in use, based on Kb USED divided by Kb AVAIL; 100% if Kb AVAIL is zero.

START/LIMIT

For device swap areas, this value is the block address on the mass storage device of the start of the swap area; except on Series 800 systems, where it is always "—", because Series 800 only swaps to a complete device or partition. The value is normally 0 for entire swap devices, or the end of the file system for devices containing both a file system and swap space.

For file system swap areas, LIMIT is the maximum number of 1-Kbyte blocks available from the file system, the same as the *limit* value given to **swapon**. A file system LIMIT value of **none** means there is no fixed limit; all space is available except that used for files, less the blocks represented by **minfree** (see *fs*(4)) plus RESERVE.

RESERVE

For device swap areas, this value is always "—". For file system swap areas, this value is the number of 1-Kbyte blocks reserved for file system use by ordinary users, the same as the *reserve* value given to **swapon**.

PRI
The same as the *priority* value given to **swapon**. This value indicates the order in which space is taken from the devices and file systems used for swapping. Space is taken from lower priority swap areas first. *priority* can have a value between 0 and (**NSWPRI** - 1) (normally 10), and has a default value of 1.

NAME
For device swap areas, the block special file name whose major and minor numbers match the swap device's ID. The *swapinfo* command searches **/dev** for device names first, then **/dev/dsk**, then **/dev/rdsk**. If no matching block special file is found, **swapinfo** prints the device ID (major and minor values), for example, **0,0x0e0200**.

For file system swap areas, NAME is the name of a directory on the file system, as given to **swapon**.

Because it needs kernel access, **swapinfo** normally succeeds only for the super-user. It reports warnings or errors if unable to access kernel resources such as **/dev/mem** or **/dev/kmem**.

Options

swapon recognizes the following options:

-m Display the AVAIL, USED, FREE, LIMIT, and RESERVE values in Mbytes instead of Kbytes, rounding off to the nearest whole Mbyte (multiples of 1024^2, compatible with the units used in **/etc/disktab**). The output header format changes from **Kb** to **Mb** accordingly.

-t Add a totals line with a TYPE of **tot**. Beware — this line might be misleading if a subset of **-dfh** is specified.

-a Show all device swap areas, including those configured into the kernel but currently unused. The word **unused** appears after the NAME, and the Kb AVAIL, Kb USED, and Kb FREE values are 0. The **-a** option is ignored unless the **-d** option is present or is true by default.

-d Print information about device swap areas only. This modifies the output header appropriately.

-f Print information about file system swap areas only. This modifies the output header appropriately.

-h Print information about swap space on hold only.

The **-d**, **-f**, and **-h** options can be combined. The default is **-dfh**.

-q Quiet mode. Print only a total Kb AVAIL value (with the **-m** option, Mb AVAIL); that is, the total swap space available on the system (device or file system swap space only if only one of **-d** or **-f** is specified), for possible use by programs that want a quick total. If **-q** is specified, the **-t** and **-a** options are ignored.

-w Print a warning about each device swap area that contains wasted space; that is, a fraction of a full swap chunk (see discussion of **Kb USED** above). This option is effective only if **-d** is also specified.

HP Clustered Environment

Client nodes can swap either locally or to another cluster node that offers the swap service. If **swapinfo** is invoked on a client node that swaps locally, local swap information is reported. If **swapinfo** is invoked on a client node that swaps remotely, it prints one line to standard output that names the swap server host. Refer to *System Administrator Manual*s for more information on swapping in an HP clustered environment.

RETURN VALUE

swapinfo returns 0 if it completes successfully (including if any warnings are issued), or 1 if it reports any errors.

DIAGNOSTICS

swapinfo prints messages to standard error if it has any problems.

EXAMPLES

List all file system swap areas with a totals line:

```
swapinfo -ft
```

AUTHOR

swapinfo was developed by HP.

SEE ALSO
swapon(1M), swapon(2), checklist(4), fs(4).

NAME

swapon - enable additional device or file system for paging and swapping

SYNOPSIS

/etc/swapon -a [**-u**]
/etc/swapon [**-p** *priority*] [**-u**] [**-e** | **-f**] *device* ...
/etc/swapon [**-m** *min*] [**-l** *limit*] [**-r** *reserve*] [**-p** *priority*] *directory*
/etc/swapon *directory* [*min limit reserve priority*]

DESCRIPTION

swapon is used to enable additional devices or file systems on which paging and swapping are to take place. By enabling a **device** for paging and swapping, the device can be accessed directly (without going through the file system) during paging and swapping activity. A device is accessed indirectly through the file system when a **file system** is enabled for paging and swapping. There are advantages and disadvantages to both.

Paging or swapping directly to a **device** is faster than doing so through the file system. However, the space on the device that is allocated to paging and swapping cannot be used for anything else, even if it is not being actively paged or swapped to.

Paging or swapping through the **file system**, while slower, provides a more efficient use of the space on the device. Space that is not being used for paging and swapping in this case can be used by the file system. Keep these tradeoffs in mind when enabling devices or file systems for paging and swapping.

The system begins by paging and swapping on only a single device so that only one disk is required at bootstrap time. Calls to **swapon** normally occur in the system multi-user initialization file **/etc/rc** making all swap space available so that the paging and swapping activity is interleaved across several disks.

Normally, the **-a** argument is given, causing all devices marked as **swap** and all file systems marked as **swapfs** in **/etc/checklist** to be made available to the swap system. By using the first field in **/etc/checklist,** (*special file name*) or (*directory*), the system determines which block device or file system to use. The *special file name* specified for each **swap** entry must specify a block special file. The *directory* specified for each **swapfs** entry must specify a directory on the file system to be enabled.

The second form of **swapon** enables individual block devices to be used for paging and swapping. The *device* name must specify a block special file. If more than one device is given, any options specified will be applied to all devices. If a file system exists on the specified block device and the **-e** or **-f** option (see Options below) is not specified, **swapon** fails and a warning message is given. This prevents a file system from being inadvertently destroyed. To force paging and swapping to a device containing a file system, the **-f** option can be used. Use this with extreme caution!

In either of the previous forms, an attempt to enable swapping to *device* will fail and a warning message will be issued if **swapon** determines the device is being used by **savecore** to retrieve system dump information. The **-u** option can be used to forcibly enable swapping to devices being used by **savecore**; however this may overwrite system dump information contained on *device* (see *savecore*(1M)).

The last two forms of **swapon** provide two different methods for enabling file systems for paging and swapping. The third form is the preferred method, with the fourth being provided only for backward compatibility. The *directory* name specifies a directory on the file system that is to be enabled for paging and swapping. The optional arguments to the fourth form have the same meaning as the arguments to the options in the third form and are described below (see Options). Note that in the fourth form, if any of the optional arguments are specified, all must be specified.

In the HP Clustered environment, client nodes can swap locally, swap to another client node acting as a swap server, or swap to the root server. Client nodes swapping locally can enable additional local devices for swapping. Swap resources for clients swapping remotely can be enabled on the root server or on the client node acting as its swap server. Refer to *System Administrator* manuals for more information on swapping in an HP Clustered environment.

Options

swapon recognizes the following options and arguments:

-a Cause all devices marked as **swap** and all file systems marked as **swapfs** in **/etc/checklist** to be made available to the swap system. The *options* field in

`/etc/checklist` entries is read by **swapon**, and must contain elements formatted as follows:

`min=`*min* See `-m` *min* below for value of *min*.

`lim=`*limit* See `-l` *limit* below for value of *limit*.

`res=`*reserve*
See `-r` *reserve* below for value of *reserve*.

`pri=`*priority*
See `-p` *priority* below for value of *priority*.

See *checklist*(4) for example entry.

`-e`
Use space after end of file system on block device for paging and swapping. This option is not supported on Series 800 only. An error message is returned if a file system is not found on the device or if this option is used on a Series 800 system. This option cannot be used with the `-f` option. Do not confuse this with swapping to a **file system**; this option is for use with a disk that has *both* file system and swap space on it.

`-f`
Normally, if a file system exists on the *device* to be enabled, **swapon** fails and outputs a warning message. This option forces the *device* to be enabled which will destroy the file system on it; use with extreme caution. This option cannot be used with the `-e` option.

`-m` *min*
min indicates the number of file system blocks the swap system will initially take from the file system. The default value for *min* is 0, indicating no swap space is to be allocated initially. Block size is defined by the administrator at the time the file system is created.

`-l` *limit*
limit specifies the maximum number of blocks the swap system is allowed to take from the disk, provided space is available that is not reserved for exclusive use by the file system. See WARNINGS below. The default value for *limit* is 0, indicating there is no limit to the amount of file system space the swap system can use. Block size is defined by the administrator at the time the file system is created.

`-r` *reserve*
reserve specifies the number of file system blocks in addition to the space currently occupied by the file system that are reserved for file system use only, making them unavailable to the swap system. See WARNINGS below. The default value for *reserve* is 0 indicating that no file system space is reserved for file system use only. Block size is defined by the administrator at the time the file system is created.

`-p` *priority*
priority indicates the order in which space is taken from the file systems and devices used for swapping. Space is taken from the lower priority systems first. *priority* can have a value between 0-10 and has a default value of 1.

`-u`
Unlock block device files which are being used by *savecore*. Normally, **swapon** will not enable swapping on *device* if it is being used by *savecore* to retrieve system dump information. The list of devices in use is maintained in file `/etc/savecore.LCK`. This option forces *device* to be enabled which may overwrite any system dump information contained on *device*. This option should be used with extreme caution, and is not currently supported on Series 300/400/700 machines.

RETURN VALUE

swapon returns one of the following values upon completion:

0	Successful completion.
>0	Error condition occurred. See *swapon*(2) for more information about error message meanings.

EXAMPLES

The first two examples enable swapping to the file system containing the `/swap` directory. The maximum number of file system blocks available to the swap system is set to 5000, the number of file system blocks reserved for file system use only is set to 10000, and the priority is set to 2. The number of file system

blocks initially taken by the swap system defaults to 0 in the first example, and is set to 0 in the second example. On a file system with an 8K block size, these examples allocate approximately 40 MB of file system swap.

```
/etc/swapon -l 5000 -r 10000 -p 2 /swap
/etc/swapon /swap 0 5000 10000 2
```

This example enables swapping to two block devices and sets the priority of both devices to 0.

```
/etc/swapon -p 0 /dev/dsk/c10d0s0 /dev/dsk/c13d0s0
```

This example enables swapping to a block device on a Series 300, 400, or 700 system, using the space after the end of the file system for swap and letting the priority default to 1.

```
/etc/swapon -e /dev/dsk/c1400d0s0
```

This example enables swapping to a block device, forcing swapping even if a file system exists on the device.

```
/etc/swapon -f /dev/dsk/c12d0s0
```

WARNINGS

The file system block size used by the `-l`, `-m`, and `-r` options varies between file systems, and is defined by the system administrator at the time the file system is created. The `dumpfs` command can be used to determine the block size for a particular file system (see *dumpfs*(1M)).

The system allocates no fewer file system blocks than the amount specified in *min*. However, to make the most efficient use of space, more than *min* blocks might be initially taken from the file system. The actual amount taken will not exceed the number of file system blocks indicated in *limit*.

Swapping to the file system is usually slower than swapping to a device.

When using the `-l` and `-r` options, the reserve space specified by the `-r` option takes precedence over the `-l` option. Thus, if:

D = total available disk space R = reserve space specified by the `-r` option *limit* = swap space specified by the `-l` option L = space currently available to the swap system F = space currently occupied by the file system

the following relationships hold:

$F + R + limit < D$	in normal operation
$L = 0$	if $F + R \geq D$
$0 \leq L \leq limit$	if $F + R + limit \geq D$

DEPENDENCIES

Series 800

The `-e` option is not supported on Series 800 systems.

Series 300, 400, 700

The `-u` option is not supported on Series 300, 400, and 700 systems.

FILES

`/dev/dsk/c#d#s#`	Normal paging devices.
`/etc/checklist`	File system table.
`/etc/savecore.LCK`	List of devices being used by `savecore`.

AUTHOR

`swapon` was developed by HP and the University of California, Berkeley.

SEE ALSO

savecore(1M), swapon(2), checklist(4).

(Requires Optional SwitchOver/UX Software)

NAME

switchdiskl - lock disks

SYNOPSIS

`switchdiskl` [`-f` *infofile*] [`-c`] *hostname*

Remarks:

This command requires installation of optional SwitchOver/UX software (not included in the standard HP-UX operating system) before it can be used.

DESCRIPTION

`switchdiskl` sets (or, with the `-c` option, clears) disk locks. These locks prevent two processors from trying to use the same disks during a SwitchOver/UX takeover, when a standby processor detects that a primary processor has failed, and reboots from the primary's disks.

Arguments

`-f` *infofile* *infofile* Specify the switch-over information file. Default is `/etc/switch/Switchinfo`.

`-c` Clear locks owned by processor executing the `switchdiskl` command.

hostname Name of the host running the `switchdiskl` command.

`switchdiskl` reads the information file and finds the section describing the given host. In that section, it looks for the `rootdisk`, `rootmir`, and `rootthird` entries. The `rootdisk` entry gives the name of the root disk used by that host. The `rootmir` entry appears only when the root is mirrored. The `rootmir` entry names the secondary half of the mirror in the case of mirroring with DataPair, and the second boot device in the case of mirroring with the Logical Volume Manager. The `rootthird` entry appears only when there is a third copy of root, and names the third boot device. This entry is only used in the case of three-way mirroring with the Logical Volume Manager. All entries use the host's name for these disks; this is in contrast to the `rootdisk#`, `rootmir#`, and `rootthird#` entries in the section describing the standby, which use the standby's names.

`switchdiskl` (without `-c` option) is run at boot-up on both primary and standby hosts. Its function in this case is to claim the disks for the running processor.

The lock is a (binary) file named `LABEL` in section 6 of the disk containing the root file system. (SwitchOver/UX requires that the root be on a disk with a boot section in section 6). The lock contains the software ID of the processor owning it. The lock is examined both when the disk is originally opened, and whenever the logical connection to the disk is broken (as a result of a reset or a power failure).

As part of the takeover procedure, the standby processor overwrites the primary's lock with its own ID, then sends a reset to the disk. This causes the primary to re-read the lock and halt in case it has not actually failed. The standby then clears its own locks using `switchdiskl -c`.

`switchdiskl -c` should also appear in each host's shutdown script, to minimize cases where a host's disks are left locked when the host is not running.

Disks locked by a given processor normally cannot be booted by another processor. If this is unacceptable for a given application, ensure that the disks are not being used then use the `hpuxboot` command with its `-F` flag to force the processor to boot regardless of the contents of the lock (see *hpuxboot*(1M)).

Only users with appropriate privileges can run `switchdiskl`.

FILES

`/etc/switch/disklock`
subprogram

`/etc/switch/Switchinfo`
default SwitchOver/UX information file

AUTHOR

`switchdiskl` was developed by HP.

SEE ALSO

switchheartb(1m), switchreadp(1m), switchsetlan(1m), switchsetflg(1m), switchinfo(4).

NAME

switchheartb - send state-of-health messages to standby

SYNOPSIS

switchheartb [**-f** *infofile*]

Remarks:

This command requires installation of optional SwitchOver/UX software (not included in the standard HP-UX operating system) before it can be used.

DESCRIPTION

switchheartb starts the SwitchOver/UX **heartbeat** daemon. **heartbeat** sends state-of-health messages to the standby host. When the standby determines that the messages have stopped, it initiates a takeover, locking the current host's disks, and rebooting as the current host. **heartbeat** also accesses the root devices periodically to ensure that the primary host notices the takeover (and halts).

Arguments

-f *infofile* Names the information file. Default is **/etc/switch/Switchinfo**.

switchheartb is run at bootup by primary hosts only, usually from **/etc/inittab**.

switchheartb finds the section in the information file describing the current host. It sends messages across each LAN link described by an entry of the form

lan# = *station address*

where # is **0-9** or **a-f**. This character is the logical unit (lu) number for the LAN card. *station address* is the station address, in hexadecimal, with a leading **0x**. The **lan#** name corresponds to the LAN device file name. For example, a **lan0** entry causes messages to be sent via **/dev/lan0**. The destination station address is named by the corresponding entry in the section describing the standby host. That is, **/dev/lan0** must address the same network on both primary and standby.

In the information file, several entries in the section for this host control how the **heartbeat** daemon runs:

rtprio Gives the real-time priority for the process (see *rtprio*(2)), or **no** if no real-time priority is desired. The default is **0**.

lockmem Tells whether the daemon should be locked into main memory (**yes**) or not (**no**). The default is **yes**.

pulserate tells how often heartbeat messages are to be sent; **pulserate = 30** means every 30 seconds. The default is 10.

logfile names the file to which **heartbeat** writes error messages.

rootdisk Names the (raw) root device of the primary which *heartbeat* periodically accesses to ensure that the current host notices the takeover and halts.

rootmir Names the second (raw) root device of the primary. This entry is required only if the root is mirrored.

rootthird Names the third (raw) root device of the primary. This entry is required only if mirrored three-way using the Logical Volume Manager.

Once the standby starts to monitor this host, any failure of **heartbeat** causes a takeover. Therefore, before killing the daemon (normally never done) or rebooting the machine, stop the **readpulse** daemon running on the standby.

Only users with appropriate privileges can run **switchheartb**.

AUTHOR

switchheartb was developed by HP.

FILES

/dev/lan* interface to LAN cards

/etc/switch/Switchinfo default SwitchOver/UX information file

`/etc/switch/writeroot`
touches root every 30 seconds
`/etc/switch/heartbeat`
heartbeat daemon

SEE ALSO
switchsetlan(1m), switchreadp(1m), switchdiskl(1m), switchsetflg(1m), switchinfo(4).

(Requires Optional SwitchOver/UX Software)

NAME

switchreadp - monitor health of primary host(s)

SYNOPSIS

switchreadp [**-f** *infofile*]

Remarks

This command requires installation of optional SwitchOver/UX software (not included in the standard HP-UX operating system) before it can be used.

DESCRIPTION

switchreadp runs the **readpulse** daemon, which monitors state-of-health messages from the primary hosts of the SwitchOver/UX group. When the daemon detects that one of the primaries has failed, it begins a takeover procedure, eventually causing the standby system processor to reboot from the failed primary's disks.

Options

-f *infofile* Names the information file. Default is **/etc/switch/Switchinfo**.

The command is run at bootup by the standby host only, usually from **/etc/inittab**.

switchreadp monitors each primary host having a **prihost#** entry in the standby's section of the information file. The # is a single character from **0-9**. The daemon listens on each lan with a **lan#** entry in this section. The numbers for these two types of entries are unrelated.

When the daemon determines that a primary has failed, it executes **/etc/switch/become**, a shell script that performs the takeover.

switchreadp examines several entries in the information file. The following entries appear in the section describing each primary:

timeout Time (in seconds) after which **readpulse** should assume that this primary has failed.

bootpath Primary boot path used when rebooting the primary host. Passed to **become**.

bootmir Boot path of the alternate boot device used for rebooting the primary host. For mirroring with DataPair, the alternate device is the secondary half of the mirrored boot device. For mirroring with Logical Volume Manager, the alternate device is a second physical volume configured as a boot device. If the root is not mirrored, this entry is omitted. Passed to **become**.

bootthird Boot path of a third boot device used for rebooting the primary host. The third boot device is a third physical volume configured as a boot device. If the root is not mirrored using Logical Volume Manager, this entry is omitted.

The following entries appear in the section describing the standby:

rtprio Gives the real-time priority for the process (see rtprio(2)), or **no** if no real-time priority is desired. The default is 0.

lockmem Tells whether the daemon should be locked into main memory (**yes**) or not (**no**). The default is **yes**.

logfile Names the file to which **readpulse** writes error messages. Messages describing the takeover also appear.

prihost# Names the primary host to monitor. # is a single character from **0-9**.

rootdisk# Names the (raw) root disk. # is a single character from **0-9**. The root named belongs to the host named in the corresponding **prihost#** entry. The name is the one used by the standby. Passed to **become**.

rootmir# Similar to **rootdisk#**, but names the second root device. If the root is not mirrored, this entry is omitted. Passed to **become**.

rootthird# Similar to **rootdisk#**, but names the third root device. If the root is not mirrored, this entry is omitted. Passed to **become**.

Only users with appropriate privileges can run `switchreadp`.

AUTHOR

`switchreadp` was developed by HP.

FILES

`/dev/lan*`	interface to LAN cards
`/etc/switch/Switchinfo`	default SwitchOver/UX information file
`/etc/switch/readpulse`	readpulse daemon
`/etc/switch/become`	performs takeover
`/etc/switch/incdate`	adjusts clock on takeover
`/etc/switch/switchdiskl`	locks disks on takeover
`/etc/switch/checkroot`	checks disks on takeover
`/etc/switch/setboot`	sets boot paths on takeover

SEE ALSO

switchsetlan(1m), switchheartb(1m), switchdiskl(1m), switchsetflg(1m), switchinfo(4).

NAME

switchsetflg - set the boot flags

SYNOPSIS

switchsetflg [**-p** | **-s**] *hostname*

Remarks:

This command requires installation of optional SwitchOver/UX software (not included in the standard HP-UX operating system) before it can be used.

DESCRIPTION

switchsetflg sets the boot flags in stable storage as appropriate for the SwitchOver host type, primary or standby. For a primary host, autoboot and autosearch are turned off. For a standby host, autoboot is turned on. **switchsetflg** is for use on SCSI configurations only. Do not use on Fiber Link configurations.

Options

-p Set the values as appropriate for a primary host.

-s Set the values as appropriate for a standby host.

If no options are specified, **switchsetflg** prints the current values of the autoboot and autosearch flags in stable storage.

switchsetflg should be executed at bootup, normally from **/etc/inittab**. **switchsetflg** is executed by each member of the SwitchOver/UX host group.

AUTHOR

switchsetflg was developed by HP.

SEE ALSO

switchheartb(1m), switchreadp(1m), switchsetlan(1m), switchdiskl(1m), switchinfo(4).

(Requires Optional SwitchOver/UX Software)

NAME

switchsetlan - set LAN station address

SYNOPSIS

switchsetlan [**-f** *infofile*] *hostname*

Remarks:

This command requires installation of optional SwitchOver/UX software (not included in the standard HP-UX operating system) before it can be used.

DESCRIPTION

switchsetlan changes the network station addresses for LAN cards on the currently running system processor to the addresses listed in the SwitchOver/UX information file.

Arguments

-f *infofile* Name of the information file. Default is **/etc/switch/Switchinfo**.

hostname Name of the host running the **switchsetlan** command.

switchsetlan finds the section in the information file describing the given host. Within this section, it finds all entries of the form

lan# = **station address**

where # is **0-9** or **a-f**, and *station address* is the new station address in hexadecimal beginning with **0x**. For each entry, it changes the station address for the LAN card attached to **/dev/lan#** to the given address. (Note: the number (or letter) at the end of the filename is the logical unit number, which is not necessarily the interface unit number used by **ifconfig** (see *ifconfig*(1M). **ioscan** shows the correspondence between hardware paths (backplane slot numbers) and logical units – see *ioscan*(1M)).

The new station addresses are supplied by HP for use with the SwitchOver/UX software product. They are not the hardware station addresses of any cards. For each host in the SwitchOver/UX group, there should be a new station address for each LAN interface.

switchsetlan should be executed at bootup, normally from **/etc/rc**, before other networking software is started. It is executed by each member of the SwitchOver/UX group.

The addresses on the networking cards are not used because they vary, depending on which system processor is running a given host. The new station addresses depend only on the host. Using these new addresses ensures that Internet software does not see an inconsistent mapping between Internet addresses and station addresses when a primary system processor fails and a standby system processor takes over.

switchsetlan is supported on Ethernet and FDDI LAN interfaces only.

Only users with appropriate privileges can run **switchsetlan**.

AUTHOR

switchsetlan was developed by HP.

FILES

/etc/switch/setlan1 subprogram

/dev/lan* interface to LAN cards

/etc/switch/Switchinfo default SwitchOver/UX information file

SEE ALSO

switchheartb(1m), switchreadp(1m), switchdiskl(1m), switchsetflg(1m), switchinfo(4).

NAME

sync - synchronize file systems

SYNOPSIS

sync [**-l**]

DESCRIPTION

sync executes the **sync()** system call (see *sync*(2)). If the system is to be stopped, the **sync** command must be called to ensure file system integrity.

sync flushes all previously unwritten system buffers including modified super blocks, modified inodes, and delayed block I/O out to disk. This ensures that all file modifications are properly saved before performing a critical operation such as a system shutdown. For additional protection from power failures or possible system crashes, use **syncer** to execute **sync** automatically at periodic intervals (see *syncer*(1M)).

Options

sync recognizes the following options:

-l (ell) Execute the **lsync()** system call instead (see *sync*(2)). If the machine is a cluster node, **sync -l** causes only the local node to be synced, while **sync** causes the entire cluster to be synced.

AUTHOR

sync was developed by AT&T and HP.

SEE ALSO

syncer(1M), sync(2).

STANDARDS CONFORMANCE

sync: SVID2

NAME

syncer - periodically sync for file system integrity

SYNOPSIS

/etc/syncer [*seconds*] [**-ls**] [**-d** *directory* ...]

DESCRIPTION

syncer is a program that periodically executes **sync()** or **lsync()** at an interval determined by the input argument *seconds* (see *sync*(2)). If *seconds* is not specified, the default interval is every 30 seconds. This ensures that the file system is fairly up-to-date in case of a crash. This command should not be executed directly, but should be executed at system boot time via **/etc/rc**, which is invoked at boot time via **/etc/inittab**.

syncer also updates the **/etc/mnttab** file if it does not match current kernel mount information.

Options

syncer recognizes the following options:

-l (ell) Cause **syncer** to use **lsync()** instead of **sync()**. **lsync** performs a local sync, whereas the **sync** performs a cluster-wide sync (see *sync*(2) for details).

-s Cause **syncer** to not update the **/etc/mnttab** file. Use of this option is provided for special cases of backward compatilibity only, and is strongly discouraged. This option may be removed in a future release.

-d Open directories for cache benefit. All directories must be specified by their full path name. If the **-d** option is not used, no directories are opened.

AUTHOR

syncer was developed by the University of California, Berkeley and HP.

SEE ALSO

brc(1M), init(1M), sync(1M), sync(2).

NAME

sysdef - analyze system definition information

SYNOPSIS

/etc/sysdef [**-n** *kernel_file*]

DESCRIPTION

sysdef analyzes the named operating system file specified by the **-n** option, and extracts configuration information. This includes device information as well tunable parameters. If no file is specified, **/hp-ux** is analyzed by default,

EXAMPLES

Analyze the **/hp-ux** kernel file:

```
sysdef
```

Analyze the operating system kernel file **/os_file**:

```
sysdef -n /os_file
```

FILES

/hp-ux default operating system file.

NAME

sysdiag - on-line diagnostic subsystem interface

SYNOPSIS

/usr/diag/bin/sysdiag

DESCRIPTION

sysdiag is the command interpreter for the online diagnostic subsystem. Its primary role is to provide a common user interface to all of the online diagnostic programs. The set of commands understood by **sysdiag** is listed below. For a complete description of each command see the *Online Diagnostics Subsystem Manual* vol. 1, *Diagnostic User Interface* chapter or type **help** [*command*] at the **sysdiag** prompt. **sysdiag** accepts commands from either standard input or files of commands.

The online diagnostic subsystem enables users to run online diagnostic programs. The online diagnostic subsystem also provides utility functions, which are listed below. All commands may be abbreviated. Many of these commands have numerous options. For a list of these and explanation of their use, type:

help [*command*]

at the **sysdiag** prompt.

Command Summary

sysdiag supports the following *command*s:

abort	Abort an active diagnostic program.
ci or **!** or **:**	Exit to the command interpreter (shell).
codetest	Used only for diagnostic development.
diagsystem	Manipulate internal diagnostic subsystem processes, usually invisible to the user.
do	Execute a command in the diagnostic history stack.
exit	Exit.
hardcopy	Send input and output to a hardcopy device.
help or **?**	Get further information about commands.
install	Add a diagnostic or utility to the online diagnostic subsystem
list	List the installed diagnostics and utilities.
listredo	Display the diagnostic history stack.
mode	Set or display the system state: single-user or multi-user.
modify	Change information about an installed diagnostic or utility.
outfile	Print input and output to a named file.
purge	Remove a diagnostic or utility from the diagnostic subsystem.
redo	Edit a command in the diagnostic history stack.
redoload	Replace the current diagnostic history stack with a previously saved one.
redosave	Save the current diagnostic history stack.
redosize	Set the maximum size to which the diagnostic history stack can grow.
resume	Resume executing a previously suspended diagnostic or utility.
run	Execute a diagnostic or utility. **run** may be implied; the name of the diagnostic or utility can be given without preceding it with **run**.
setvar	Explicitly set environmental variables used by the online diagnostic subsystem.
showactive	List all running, suspended, and aborting diagnostic programs and utilities.
showstate	Display the system mode (single-user or multi-user) and the user's diagnostic security capability.

`suspend`	Suspend execution of a diagnostic program or utility.
`unlock`	Explicitly unlock a device.
`usefile`	Instructs the diagnostic subsystem to take all further input from the named file.
`wait`	Causes the diagnostic subsystem to wait for background processes to complete, before accepting more input.

AUTHOR

`sysdiag` was developed by HP.

FILES

`/usr/diag/bin/*`	diagnostic programs
`/usr/diag/bin/security`	access list for diagnostic users
`/dev/diag/*`	diagnostic special files

SEE ALSO

Online Diagnostics Subsystem Manual, Volume 1.

NAME

syslogd - log systems messages

SYNOPSIS

/etc/syslogd [**-f** *configfile*] [**-m** *markinterval*] [**-d**]

DESCRIPTION

syslogd reads and logs messages into a set of files described by the configuration file **/etc/syslog.conf**.

Only the super-user can run **syslogd**.

Options

syslogd recognizes the following options:

-f *configfile*	Use *configfile* instead of **/etc/syslog.conf**.
-m *markinterval*	Wait *markinterval* minutes between mark messages, instead of 20 minutes.
-d	Turn on debugging.

syslogd creates the file **/etc/syslog.pid**, if possible, containing a single line with its process ID. This can be used to kill or reconfigure **syslogd.**

To kill **syslogd**, send it a terminate signal:

```
kill 'cat /etc/syslog.pid'
```

To make **syslogd**, re-read its configuration file, send it a **HANGUP** signal:

```
kill -HUP 'cat /etc/syslog.pid'
```

syslogd collects messages from the UNIX domain socket **/dev/log.un**, an Internet domain socket specified in **/etc/services**, and from the named pipe **/dev/log**. By default, local programs calling **syslog()** send log messages to the UNIX domain socket (see *syslog*(3C)). If UNIX domain sockets are not configured on the system, they write to the named pipe instead. If INET domain sockets are not configured, **syslogd** does not receive messages forwarded from other hosts, nor does it forward messages (see below).

Each message is one line. A message can contain a priority code, marked by a number in angle braces at the beginning of the line. Priorities are defined in the header file **<syslog.h>**.

syslogd configures itself when it starts up and whenever it receives a hangup signal. Lines in the configuration file consist of a *selector* to determine the message priorities to which the line applies and an *action*. The *action* field is separated from the selector by one or more tabs.

Selectors are semicolon separated lists of priority specifiers. Each priority has a *facility* indicating the subsystem that generated the message, a dot, and a *level* indicating the severity of the message. Symbolic names can be used. An asterisk selects all facilities. All messages of the specified level or higher (greater severity) are selected. More than one facility can be selected, using commas to separate them. For example:

```
*.emerg;mail,daemon.crit
```

selects all facilities at the **emerg** level and the **mail** and **daemon** facilities at the **crit** level.

Known facilities and levels recognized by **syslogd** are those listed in *syslog*(3C) converted to lowercase without the leading **LOG_**. The additional facility **mark** has a message at priority LOG_INFO sent to it every 20 minutes (this can be changed by using the **-m** flag). The **mark** facility is not enabled by a facility field containing an asterisk. The level **none** can be used to disable a particular facility. For example,

```
*.debug;mail.none
```

selects all messages except **mail** messages.

The second part of each line describes where the message is to be logged if this line is selected. There are four forms:

- A filename (beginning with a leading slash). The file is opened in append mode. If the file does not exist, it is not created.

- A hostname preceded by an `@` character. Selected messages are forwarded to the **syslogd** on the named host.
- A comma-separated list of users. Selected messages are written to those users' terminals if they are logged in.
- An asterisk. Selected messages are written to the terminals of all logged-in users.

Blank lines and lines beginning with a `#` character are ignored.

For example, the configuration file:

```
mark.debug                  /dev/console
mail.debug                  /usr/spool/mqueue/syslog
*.info;mail.none            /usr/adm/syslog
*.alert                     /dev/console
*.alert                     root,eric,kridle
*.emerg                     *
*.emerg                     @admin
```

logs 20 minute marks onto the system console, all mail system messages to **/usr/spool/mqueue/syslog**, and all messages at **info** and above, except mail messages, to the file **/usr/adm/syslog**. Messages at **alert** and above are logged to the console and to the users **root**, **eric**, and **kridle** if they are logged in. **emerg** messages are written to all logged-in users' terminals, and forwarded to the host **admin**.

WARNINGS

A configuration file selector selects all messages at the specified level *or higher*. The configuration lines:

```
user.debug      /tmp/logfile
user.info       /tmp/logfile
```

cause the logfile to get *two* copies of all **user** messages at level **info** and above.

HP-UX does not support kernel logging through the special log device **/dev/klog**.

AUTHOR

syslogd was developed by the University of California, Berkeley.

FILES

/etc/syslog.conf
: configuration file

/etc/syslog.pid
: process ID

/dev/log.un
: the UNIX domain socket on which **syslogd** reads log messages

/dev/log
: the named pipe on which **syslogd** reads log messages

SEE ALSO

logger(1), syslog(3C).

NAME

telnetd - TELNET protocol server

SYNOPSIS

/etc/telnetd [**-b** [*bannerfile*]]

DESCRIPTION

telnetd is a server that supports the DARPA standard TELNET virtual terminal protocol. The Internet daemon (**inetd**) executes **telnetd** when it receives a service request at the port listed in the services data base for **telnet** using the **tcp** protocol (see *inetd*(1M) and *services*(4)).

telnetd operates by allocating a pseudo-terminal device (see *pty*(7)) for a client, then creating a login process which has the slave side of the pseudo-terminal as **stdin**, **stdout**, and **stderr**. **telnetd** manipulates the master side of the pseudo-terminal, implementing the TELNET protocol and passing characters between the client and login process.

When a TELNET session is started up, **telnetd** sends TELNET options to the client side indicating a willingness to do **remote echo** of characters, to *suppress go ahead*, and to receive *terminal type information* from the remote client. If the remote client is willing, the remote terminal type is propagated in the environment of the created login process. The pseudo-terminal allocated to the client is configured as a normal terminal is for login, with the exception of echoing characters (see *tty*(7)).

telnetd is willing to *do*: *echo*, *binary*, *suppress go ahead* , and *timing mark* .
telnetd is willing to have the remote client *do*: *binary*, *terminal type* , and *suppress go ahead* .

To start **telnetd** from the Internet daemon, the configuration file **/etc/inetd.conf** must contain an entry as follows:

```
telnet stream tcp nowait root /etc/telnetd telnetd
```

To override the standard **telnetd** login banner, specify a file containing a custom banner with the **-b** *bannerfile* option. For example, to use **/etc/issue** as the login banner, have **inetd** start **telnetd** with the following line in **/etc/inetd.conf**:

```
telnet stream tcp nowait root /etc/telnetd telnetd -b/etc/issue
```

If *bannerfile* is not specified, **telnetd** does not print a login banner.

DIAGNOSTICS

If any error is encountered by **telnetd** in establishing the connection, an error message is returned through the connection after which the connection is closed and the server exits. Any errors generated by the login process or its descendents are passed through as ordinary data.

All ptys on remote host in use
The server was unable to obtain a pseudo-terminal for use with the login process. Either all pseudo-terminals were in use or the pty driver has not been properly set up (see *pty*(7)).

Next step: Check the **pty** configuration of the host where **telnetd** is executing.

fork: No more processes
telnetd was unable to fork a process to handle the incoming connection.

Next step: Wait a period of time and try again. If this message persists, the server's host may have runaway processes that are using all the entries in the process table.

/bin/login: ...
The login program could not be started via **exec( )** for the reason indicated (see *exec*(2).

WARNINGS

The terminal type name received from the remote client is converted to lowercase.

telnetd never sends TELNET *go ahead* commands.

AUTHOR

telnetd was developed by the University of California, Berkeley.

SEE ALSO

login(1), rlogin(1), telnet(1), inetd(1M), pty(7), tty(7), hosts(4), inetd.conf(4), inetd.sec(4), services(4).

DOD MIL_STD 1782,

RFC 854 for the TELNET protocol specification.

NAME

tftpd - trivial file transfer protocol server

SYNOPSIS

/etc/tftpd [**-R** *retransmission-timeout*] [**-T** *total-transmission-timeout*]

DESCRIPTION

tftpd is a server that supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the **tftp** service description (see *services*(4)). The server is normally started by **inetd** using the **inetd.conf** file (see *inetd*(1M) and *inetd.conf*(4)).

The **-R** option specifies the per-packet retransmission timeout, in seconds. The **-T** option specfies the total retransmission timeout, in seconds. The defaults values are 5 and 25 seconds, respectively.

tftpd requires an entry in the *passwd*(4) database for an account named **tftp**. The password field should be *, the group membership should be **guest**, and the login shell should be **/bin/false**. For example:

```
tftp:*:510:guest:tftp server:/usr/tftpdir:/bin/false
```

tftpd does a **chroot()** to the home directory of this user, thus it restricts **tftp** clients only to those files placed there (see *chroot*(2)). Furthermore, files in this directory can be read by **tftp** clients only if they are readable by the user **tftp**. Files can be written only if they already exist and are writable by the user **tftp**.

DIAGNOSTICS

The following diagnostics are logged to the **syslogd** daemon facility at the **error** log level.

User tftp unknown
The user **tftp** was not found in the password database, **/etc/passwd**.

Next step: Add or correct the entry for the user **tftp** in the password database **/etc/passwd**.

Unknown option <*option*> **ignored**
An invalid option was specified in the **tftpd** arguments in the **inetd** configuration file **/etc/inetd.conf**.

Next step: Remove or correct the option and reconfigure **inetd** with the command **inetd -c**.

Invalid total timeout <timeout>
The value given for the **-T** option was either not a number or was a negative number.

Next step: Correct the value given for the **-T** option and reconfigure **inetd** with the command **inetd -c**.

Invalid retransmission timeout <timeout>
The value given for the **-R** option was either not a number or was a negative number.

Next step: Correct the value given for the **-R** option and reconfigure **inetd** with the command **inetd -c**.

<system call>: ...
The system call failed; see the corresponding manual entry for a description of the system call. The reason for the failure is explained in the error message appended to the system call name.

WARNINGS

Because **tftpd** performs a **chroot()**, it cannot follow symbolic links that refer to paths outside of the home directory of the pseudo-user **tftp**.

AUTHOR

tftpd was developed by the University of California, Berkeley and HP.

SEE ALSO

tftp(1), inetd(1m), chroot(2), passwd(4).

NAME

tic - terminfo compiler

SYNOPSIS

tic [**-v**[*n*]] *file* ...

DESCRIPTION

tic translates terminfo files from source format into the compiled format. Results are placed in the directory **/usr/lib/terminfo**.

The **-v** (verbose) option causes **tic** to output trace information showing its progress. If the optional integer *n* is appended, the level of verbosity can be increased.

tic compiles all terminfo descriptions in the given files. When a **use=** field is discovered, **tic** searches first the current file, then the master file which is **./terminfo.src**.

If the environment variable **TERMINFO** is set, the results are placed in the location specified by **TERMINFO** instead of in **/usr/lib/terminfo**.

Limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

FILES

/usr/lib/terminfo/?/* compiled terminal capability data base

SEE ALSO

untic(1M), curses(3X), terminfo(4).

BUGS

Instead of searching **./terminfo.src**, **tic** should check for an existing compiled entry.

STANDARDS CONFORMANCE

tic: SVID2

NAME

tsm.lpadmin - add or remove a printer for use with *tsm*(1)

SYNOPSIS

/usr/tsm/bin/tsm.lpadmin -p I printer **-m** *model*
/usr/tsm/bin/tsm.lpadmin -x *printer*

DESCRIPTION

tsm.lpadmin is used to add (or remove) a printer to the LP spooling system when the printer is connected to the system through a terminal running the Terminal Session Manager (see *tsm*(1)). **tsm.lpadmin** is a shell script that uses **lpadmin** in the normal way but also creates a named pipe to which LP output is directed (see *lpadmin*(1)). This named pipe is opened by TSM and data flowing from it is sent to the printer through the terminal. A printer configured with **tsm.lpadmin** can be connected to a cnode (through a terminal) even though the LP spooler runs on the cluster server.

Options

tsm.lpadmin recognizes the following options:

-p *printer* Names a printer to be created with an associated pipe. If **-p** is used, **-m** must also be specified.

-m *model* Selects a model interface program for *printer*. *model* is one of the model interface names supplied with the LP software (see the Models topic in the *lpadmin*(1)) manual entry. If **-m** is used, **-p** must also be specified.

-x *printer* Removes *printer* from the LP system. No other options are allowed with **-x**.

Restrictions

To use **tsm.lpadmin** you must be user **lp** or **root**, and you must be executing on a stand-alone system or on the cluster server if in an HP Clustered Environment.

AUTHOR

tsm.lpadmin was developed by HP.

FILES

/usr/spool/lp/tsm.pipes/*

SEE ALSO

lpadmin(3M), tsm(1).

NAME

tunefs - tune up an existing file system

SYNOPSIS

/etc/tunefs [**-A**] [**-v**] [**-a** *maxcontig*] [**-d** *rotdelay*] [**-e** *maxbpg*] [**-m** *minfree*] *special-device*

DESCRIPTION

tunefs is used to alter dynamic parameters that affect file system layout policies. Parameters to be altered are specified by the options and arguments provided on the command line as described below.

tunefs affects how the file system blocks are laid out on the disk. Default parameters used by the **newfs** command (see *newfs(1m)*) cause one block to be written and a minimum of 4 milliseconds (default *rotdelay* value) to be inserted before the next one is written. For many disks, this provides optimal throughput.

More sophisticated disk controllers sometimes have read-ahead or write-caching features that allow more aggressive tuning to be used. Changing to 0 ms rotdelay allows file system blocks to be written and read consecutively. On drives that have read-ahead but no write-caching capabilities, write performance will decrease due to latencies incurred after each write.

Options

tunefs recognizes the following options and command-line arguments:

-a *maxcontig* Set the maximum number of contiguous blocks that will be laid out before forcing a rotational delay to *maxcontig* (see **-d** below). The default value is **1**, because most device drivers require one interrupt per disk transfer. For device drivers that can chain several buffers together in a single transfer, set *maxcontig* to the maximum chain length.

-d *rotdelay* *rotdelay* is the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to determine how much rotational spacing to place between successive blocks in a file.

-e *maxbpg* *maxbpg* specifies 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. Typically this value is set to about one fourth of the total blocks in a cylinder group. The intent is to prevent any single file from using up all the blocks in a single cylinder group, thus degrading access times for all files subsequently allocated in that cylinder group. The effect of this limit is to cause large files to do long seeks more frequently than if they were allowed to allocate all the blocks in a cylinder group before seeking elsewhere. For file systems with exclusively large files, this parameter should be set higher.

-m *minfree* *minfree* specifies the percentage of space that is not available to normal users; i.e., the minimum free space threshold. The default value used is 10%. This value can be set to zero. If set to zero, throughput performance drops to as little as one-third of the efficiency expected when the threshold is set at 10%. Note that if *minfree* is raised above the current usage level, users cannot allocate files until enough files have been deleted to meet the new threshold requirement.

-v (visual) Display current values contained in the primary super-block to standard output.

-A (all) Modify redundant super-blocks as well as the primary super-block as stipulated by the configuration options and arguments.

special-device *special-device* is the name of the file system to be tuned. It is either a block or character special file for an unmounted volume or volume section.

WARNINGS

tunefs should be used only on unmounted file systems (when a file system is mounted, the super-block is copied into a buffer cache, then written back to the disk when the file system is unmounted, thus destroying any changes made to the disk super-block by **tunefs** while the filesystem was mounted).

Root file system tuning is normally done during initial system software installation. Tuning the root file system after installation has little useful effect because so many files have already been written. Tuning

should be done in the install process; On Series 300/400 systems, when given the option to change file system parameters, enter 0 ms rotdelay instead of the default 4 ms. Series 700 systems select optimal tuning automatically during install.

You can tune a file system, but you can't tune a fish.

DEPENDENCIES

Series 700

In general, Series 700 systems determine optimal tuning automatically and use it during the install process and during file-system generation, making use of `tunefs` unnecessary.

AUTHOR

`tunefs` was developed by the University of California, Berkeley.

SEE ALSO

dumpfs(1M), mkfs(1M), newfs(1M), fs(4).

NAME

untic - terminfo de-compiler

SYNOPSIS

`untic` [*term*] [`-f` *file*]

DESCRIPTION

`untic` translates a terminfo file from the compiled format into the source format. If the environment variable `TERMINFO` is set to a path name, `untic` checks for a compiled terminfo description of the terminal under the path specified by `TERMINFO` before checking `/usr/lib/terminfo`. Otherwise, only `/usr/lib/terminfo` is checked.

Normally `untic` uses the terminal type obtained from the `TERM` environment variable. With the *term* (terminal type) argument, however, the user can specify the terminal type used.

With the *file* argument the user can specify the file used for translation. This option bypasses the use of the `TERM` and `TERMINFO` environment variables.

`untic` sends the de-compiled terminfo description result to standard output.

AUTHOR

`untic` was developed by HP.

FILES

`/usr/lib/terminfo/?/*` compiled terminal capability data base

SEE ALSO

tic(1M), curses(3X), terminfo(4).

NAME

update, updist - update or install HP-UX files (software products)

SYNOPSIS

[**UPDATESCRIPT='**[*filename*]**'**]
/etc/update [**-mF**] [**-s** *source*] [**-S** *series*] [**-P** *port*] [**-C** *codeword*] [**-d** *destination*] [**-r** [*kernel_gen_file*]] [**-b** *boot_partition*] [**-f** *file*] [*fileset*...]
/etc/update -i
/etc/update -c [**-m**] [**-s** *source*] [**-S** *series*] [**-P** *port*] [**-C** *codeword*]

[**UPDISTSCRIPT='**[*filename*]**'**]
/etc/updist (same except **-r** and **-b** are not supported)

/etc/mk_cnodekern *linkset_flag_file cnode_name*

DESCRIPTION

update is used interactively or non-interactively to:

- Update the HP-UX operating system and core product files,
- Install new HP-UX application software (optional products),
- Update existing optional HP-UX application software.

The **updist** command is similar to **update**, except that it installs or updates the HP-UX system or application files as "fileset packages" in a special directory. This allows the system to be a network file distribution (netdist) server. The **netdistd** network server daemon finds the files in this special directory and supplies them to a remote **update** process on request (see *netdistd*(1M)).

The **mk_cnodekern** script is used by **update** to rebuild cluster node kernels.

If no options or fileset names are specified, **update** and **updist** run interactively, providing help screens to aid in selection of installation or update options. To set default values for interactive sessions, use an "update script" (see Update Scripts below). If one or more options or fileset names are specified, **update** and **updist** run non-interactively as indicated by the options and fileset names given.

update and **updist** load files from three types of update media:

- **tar**-format serial media, normally nine-track, cartridge, or DDS tapes, or an ordinary file containing an appropriate **tar** image
- systems configured to be netdist servers
- directories, normally mounted CD-ROMs

Tape update media consist of simple **tar** archives containing product files and directories (see *tar*(1)), plus a few leading information files, and specially-crafted file paths that allow files to be grouped into filesets (see Filesets and Partitions below; also *update*(4)). When run non-interactively, **update** and **updist** run unattended and therefore do not allow loading from multiple media units (tapes). If an attempt is made to load from multiple media non-interactively, **update** refuses to begin loading; **updist** loads from the first media unit only, then terminates with a warning message.

When updating from a netdist server system or from a CD-ROM (directory), the update "media" consists of a collection of files in a directory hierarchy plus various information files that, together, act as a single media volume. (The netdist server and CD-ROM directory formats are different.)

The cdfs file system must be configured in the kernel to mount a CD-ROM.

Before loading any filesets, **update** and **updist** calculate the additional disk space consumed by the installation or update. This prevents loading of filesets when sufficient disk space is not available.

Only the super-user can run **update** and **updist**.

Options

update and **updist** support the following options when used non-interactively:

-m (match) Load new versions of all filesets currently installed on the system. For **update** the **/etc/filesets** directory is used to determine the currently loaded set. For **updist** the destination directory (netdist tree) is used. If fileset names change between releases of the HP-UX core functionality, **update** and **updist** use an internal table to

convert old names to corresponding names in the new release.

In conjunction with `-c`, `-m` causes the table of contents to show which filesets would be matched and loaded.

(In previous releases, `updist -m` matched `/etc/filesets`, not the netdist tree.)

`-F` Force non-interactive loading to proceed even if certain conditions are encountered that normally would be errors. These conditions would all be warnings if the program were run interactively. They include:

- Fileset version older on update media than already loaded on system.
- Disk space short (invades minfree) but sufficient on one or more volumes.
- Swap space insufficient or cannot be determined (S800 only).

`-s` *source*
Specify a non-default source for the update. *source* can be:

- Absolute path name of local special file representing a nine-track, cartridge, or DDS tape.
- Absolute path name of regular file containing an update media image in `tar` format.
- Hostname of a netdist server system running the `netdistd` daemon.
- Absolute path name of directory, normally a CD-ROM file system (must be listed in the checklist file or already mounted).

The default *source* is `/dev/update.src` on Series 300, 400, and 700 systems and `/dev/rmt/0m` on Series 800 systems.

`-S` *series*
Specify Series 300/400 (`-S300`), Series 700 (`-S700`), or Series 800 (`-S800`) for the type of files you expect to extract from the source media. The default is the type of system on which `update` or `updist` is run. Use this option when:

- Loading files for the non-default system type from a netdist server system.
- Using `updist` to load a mixed cluster or a netdist tree using update media that is made for more than one type of system. (Beginning at HP-UX Release 8.05, media can be made for all series of HP 9000 computers. Older media for Series 800 systems can be loaded on Series 700 systems too, although in most cases this is not advisable.)

The value of the `-S` option is ignored except when it is needed for either of the two above purposes.

`-P` *port*
Set the port number for the netdist service (applies only if the source is a netdist server). This option overrides the number in the network services database (see FILES below). It is useful when a netdist server system offers various software packages through different netdist servers at different port numbers.

`-C` *codeword*
Specify a *codeword* for allowing access to protected software loaded from a directory (normally a mounted CD-ROM). A *codeword* is a string of 16 or 29 characters. *codeword*s are issued by HP and are based on *hardware IDs* associated with specific hardware units attached to the system when the software is updated.

Hardware units that provide *hardware IDs* can be any of:

- An **SPU ID**: provides a *machine ID number* from the `uname()` system call as found in a LAN card (Series 300/400; for Release 7.0, the `lla` driver must be in the kernel), or the `SW_ID` field in Stable Storage (Series 700 and 800).
- An **HP 46084A** HP-HIL ID module or **HP 460841** HP-HIL exchange ID module: provides a *"security number"* (on Series 800 systems, the `hil` driver must be in the kernel).
- An **HP-C1707A** CD-ROM drive: provides a *security number* (the drive must be mounted, or listed in the checklist file and mountable). Not supported on Series 700 systems.

The `-C` option is required when the source is a directory, and ignored otherwise. Run `update` or `updist` interactively to set up a products list file for the directory (see FILES below) before using this

option non-interactively.

update and **updist** remember the last valid *codeword* specified in any previous invocation (see FILES below). If the same *codeword* is to be reused, the **-C** option can be omitted.

-d *destination*
Specify a non-default destination directory under which filesets are unpacked. For **update** the default destination is **/**. Some filesets *require* that the destination directory be **/**. This option is used for updating an application that can be installed anywhere on the file system.

For **updist** the default destination is **/netdist**. This option is used for creating alternate source trees for **netdistd**.

-r [*kernel_gen_file*]
(**update** only) Reboot after loading all requested filesets, if required, and (for Series 800 only) specify the kernel generation file that reflects the currently-running kernel. With this option, if any fileset is marked as requiring a system reboot, **update** reboots the system after loading files and rebuilding the kernel, regardless of currently running processes. This option is required if a fileset is selected that modifies the kernel.

On Series 300, 400, and 700 systems, no *kernel_gen_file* option argument is required or allowed, because **update** extracts a new kernel generation file from the currently running system when necessary (if and only if the **KERN-BLD** fileset is selected for loading). On Series 800 systems, *kernel_gen_file* is typically **/etc/conf/gen/S800**. There is no default value. The option argument is required because **update** must inspect or alter the generation file to succeed.

-b *boot_partition*
(**update** only; Series 800 only) Specify the current boot partition on the disk. The disk partition, not to be confused with a logical (fileset) partition, is the name of a special file relative to **/dev/rdsk**. The default is **c0d0s6**. The disk partition must contain a LIF volume.

When run interactively, **update** uses the default value, and prompts for an alternate value only if **/dev/rdsk/c0d0s6** is unusable or that partition does not contain a LIF volume.

-f *file*
Read from the specified file, rather than from the command line, the list of filesets or partitions to be loaded (see Filesets and Partitions below). Blank lines and comments in the file are ignored. Comments are remainders of lines beginning with **#** following whitespace or at the start of a line.

-i
Produce a list of *hardware IDs* available on the system for requesting a *codeword* to allow access to protected software (see the **-C** option above). Note that the **-i** option only reports on CD-ROM drives already mounted; not those that would be mounted automatically by **update** or **updist** before loading from them.

-c
Produce a table of contents from the source media. **update** and **updist** write a list of *partition* **.** *fileset* names to standard output, one name per line (see Filesets and Partitions below). The output includes comments describing the size of each fileset, its media unit number, any associated fileset flags (see *update*(4)), its version number, and its description. The output is usable as input to the **-f** option. Comment out or delete the lines for any filesets you do not want to load.

The **-m** and **-f** options cannot be used when naming filesets explicitly, and vice-versa.

Filesets and Partitions

update and **updist** load units called "filesets" that are groups of related files. One or more filesets can be further grouped into logical "partitions". Filesets and partitions are the items that you can choose from when updating.

One or more filesets or partitions can be selected for loading. The format of partition/fileset names is:

partition **.** *fileset*

The following shorthand specifications are allowed:

partition Select all filesets in a partition.

fileset Select an individual fileset.

* Select all filesets on the media. The * must be escaped in the command line to prevent its expansion by the shell.

Any fileset might require other filesets on which it depends. The update media includes this dependency information. Thus, selection of a fileset with dependencies (interactively or from the command line) causes automatic selection of other required filesets if they are not already present on the system (or for **updist**, in the netdist source tree) with a version number equal to or greater than that required. (Version numbers are stored in per-fileset index files; see FILES below.) Likewise, interactive unselection of a fileset causes automatic unselection of its other required filesets, unless you explicitly selected them or selected other filesets that require them.

update creates or rewrites fileset lists in "fileset files" under the filesets directory (see FILES below). Each file in this directory has the same name as the fileset it represents. Each line in a filesets file is the full, absolute path name (actual destination) of a corresponding file loaded as part of the fileset.

Filesets marked as requiring rebooting are always loaded first so **update** can rebuild the kernel after loading them. If rebuilding fails, **update** aborts rather than loading any other filesets, because they might include executable files that cannot run correctly on the current system version.

Updating Netdist Server Master Files

updist transfers filesets from update media to a special tree under the local file system of an **update** server system and prepares the filesets for use by the **netdistd** server process. **updist** differs from **update** as follows:

- The default destination is **/netdist**.
- Fileset files are not created.
- The kernel is not rebuilt and the system is not rebooted.
- Customize scripts are not run.
- Other information files are prepared under the special tree.

Mounted Volumes and Links

update and **updist** avoid loading files under directories that are on read-only file systems or those normally used as NFS mount points. Both commands begin by executing **mount -a** with errors ignored. They then check and require that all disks listed in **/etc/checklist** are indeed mounted (the disks must be listed in **/etc/mnttab**).

update and **updist** break hard links when re-creating updated files. They also follow symbolic links and update the targets of the symbolic links. **update** records in fileset files the absolute path of updated files, after resolving any symbolic links.

Update Scripts

An update script is a shell script that captures the program status of an interactive session. This is helpful if you want to run **update** or **updist** non-interactively later (for example, via **remsh** on a different system, or possibly repeated on the same system). Saved values from a previous interactive session also provide an optional set of default values for the next time you run **update** or **updist** interactively, as described below. The update script also serves as an added audit trail.

The update script is created or rewritten just before confirmation is requested to begin loading, each time that that point is reached (if confirmation is denied). **update** writes to the **/tmp/update.script** file and **updist** writes to **/tmp/updist.script**.

Note, some conditions that cause warnings when running interactively cause errors when running non-interactively. Also, the update script is not guaranteed to run successfully on a remote system where the context is different, especially if it is of a different Series type.

If the **UPDATESCRIPT** (for **updist**, **UPDISTSCRIPT**) environment variable is set, **update** or **updist** attempts to read the file named in the variable as an update script (shell script) in a specific format emitted by an earlier invocation of the same program. For example:

```
UPDATESCRIPT='/tmp/update.script' /etc/update
```

If the variable value is null, the program reads the appropriate default update script name. For example, the following causes **updist** to read **/tmp/updist.script**:

```
UPDISTSCRIPT='' /etc/updist
```

In the unlikely event you want to create an update script by hand, the update script format is documented in *update*(4).

If the program is able to "read back" the update script correctly, it uses the values found there as default values for selected environment variables, the update source and destination and related values, the target Series type, kernel building information, and selected fileset names.

If the previous invocation of **update** or **updist** involved the "Select Only Filesets Currently on your System" choice, and no changes were then made to the auto-selected filesets, the update script file contains the **-m** option and no fileset names. If on the other hand the update script does contain a list of fileset names, they are automatically pre-selected after readback. However, this only occurs if and when the first selection option chosen on the main screen is **Select/View Partitions and Filesets**. Also, only listed filesets found on the current update media are selected this way.

HP Clustered Environment

update and **updist** recognize filesets as containing Series 300/400, Series 700, or Series 800 files. If a fileset contains Series 300/400 files and is being loaded onto a Series 700 or Series 800 cluster server file system in a mixed cluster, **update** creates context-dependent file (CDF) elements as required (specified by the media), and records their explicit path names in fileset files. **updist** maintains separate subdirectories for each Series.

update does not install cluster servers or convert stand-alone systems to cluster servers. Use **sam** for that purpose (see *sam*(1M)).

On a clustered system, only the cluster server's checklist file is checked to ensure that all volumes are mounted. Locally mounted file systems are ignored; ensure they are properly mounted or unmounted as required prior to doing the update.

On a mixed cluster such as a Series 700 server with Series 300 client nodes, some files are shared (common); that is, not CDFs. **update** skips loading common files if they are in a fileset whose version on the update media is lower than the highest version of that fileset on the system for any architecture type (usually the opposite type). For example, suppose your Series 700 server is running 8.05 software, and you are updating your Series 300 clients to version 8.0. **update** does not load the common files in 8.0 version filesets because higher versions of those files already exist on the server. The higher version of each file is designed to be backward compatible.

update provides a way to override the default behavior. Setting the environment variable UPDATENOSKIP (with any value) causes **update** to load all common files. This might create incompatibilities between the server and clients. For example, type:

```
UPDATENOSKIP="" update
```

The mk_cnodekern Script

This script directs the building of a kernel for a cluster node if one is not already built. It is called by each Series 300 cnode's temporary copy of **/etc/inittab** when that cnode is initially booted following an update that has rebuilt the cluster server's kernel. It always restores the cnode's original **/etc/inittab** file regardless of the success or failure of the kernel build attempt.

The first argument, *linkset_flag_file*, is a value unique to the cnode in question (such as the inode number of its current kernel file) to be used to distinguish a lock file and other temporary files. The second argument, *cnode_name*, is the name of the kernel to build.

RETURN VALUE

Upon completion, **update** returns one of the following values:

- **0** the update ran successfully to completion
- **1** an error occurred and no files were loaded
- **2** an error occurred and some files might have been loaded; review the log file for details (see FILES below)

DIAGNOSTICS

Messages are displayed during interactive execution, and error messages result from invalid non-interactive invocation. For information about any failures encountered while loading filesets in either mode, inspect the log file (see FILES below).

EXAMPLES

Update or install the filesets **AMERICAN** and **NETIPC** on the system using the default source device and destination directory:

```
update AMERICAN NETIPC
```

Update or install all filesets on the media in the default source device under the directory **/tmp**:

```
update -d /tmp '*'
```

Print the contents of the update media accessed through special file **/dev/rmt**:

```
update -c -s /dev/rmt
```

Update all files in filesets currently on the system (match), rebuilding the kernel and rebooting when done if necessary:

```
update -mr
```

Extract fileset **TEXT-FMT** from the default source device and make it available to other systems via **netdistd**. In this example, **/pseudoroot** is the directory from which **netdistd** draws its files:

```
updist -d /pseudoroot TEXT-FMT
```

WARNINGS

Attempting to load from 8.0 or later update media with a 7.0 version of **update** results in an arcane message about an improper file size line in the **INFO** file. If you encounter this condition, load the new version of the **TOOL** fileset from the new update media, following the directions in the *Installing and Updating HP-UX* manual.

Always review the log file after an update for relevant errors, warnings, and other messages.

update can rebuild the kernel (including cluster node kernels, if any) and reboot the system as part of an update. Flags associated with each fileset indicate the necessity for a rebuild and reboot. In interactive mode, **update** warns you if such a fileset is selected. If the system is not in a quiescent state, you can proceed, or abort **update** and take appropriate action to bring the system to single-user state. In command-line invocation with a rebooting fileset selected, **update** aborts before loading any filesets unless the **-r** option is specified.

If any loaded fileset is flagged for reboot, and any other loaded fileset is not, **update** reboots the system twice. The first reboot brings up the new kernel and allows deferred customize scripts to run. The second reboot restarts the system in its normal state after all customization is complete.

(In this case, **update** leaves a file named **/tmp/customize** on the system. This is the master customize script that invokes all others and causes the second system reboot. It can be removed without risk.)

Most filesets are flagged such that **update** does not remove their files first (using **rmfn** — see *rmfn*(1M)). If a fileset is updated to a different destination than where currently installed, the old version might not be deleted. In this case, you might want to run **rmfn** manually before the update (before the fileset file is revised). Also, if a fileset name changes, **update** removes the fileset using its current name, which might be ineffective.

The **rmfn** command cannot be used to remove packages loaded by **updist**. To remove **updist**-package files, run **rm -r** on the corresponding package directories under the netdist source tree, then edit the central package definition file to comment out or delete the references to the package.

Fileset name translation with the **-m** option is limited to filesets in the core HP-UX product, and excludes optional or third-party products and software subsystems.

When run interactively, **update** and **updist** depend on the **TERM** environment variable to determine the display type. If the variable is absent or has the wrong value, the display might behave oddly.

Interactive **update** and **updist** use special function keys (SFKs) extensively. They do not save or restore user function-key definitions after the update is complete.

CD-ROM File System (CDFS) support must be configured into the kernel (using **sam**) before attempting to mount and load from a CD-ROM.

DEPENDENCIES

Series 700
For HP-UX releases before 8.05, such as 8.0 and 8.01, **update** and **updist** do not distinguish Series 700 and Series 800 update media. They refer to Series 700 systems and update media as "S800". In cases where the filesets on the media do not run equally well on both series, be sure to load from the correct media according to the printed label, or, in the case of a netdist server, from the correct port number. (See *netdistd*(1M) DEPENDENCIES.

NFS
update and **updist** refuse to update files on remote systems over NFS network connections. They premount all normally mounted volumes, including NFS mounts, to ensure that such files are detected and that unwanted files are not deposited under the mount point on the local disk. **update** and **updist** complete loading all local files, give warnings about remote files not loaded, and return 0 if no other problems are detected.

Networking
To load files from a netdist server system, networking services must be configured into the local kernel and turned on.

AUTHOR
update and **updist** were developed by HP.

FILES
/dev/update.src	default source for Series 300, 400, and 700 systems
/dev/rmt/0m	default source for Series 800 systems
/	default destination for **update**
/netdist	default destination for **updist** and source for **netdistd**
/etc/filesets	directory where fileset files are stored
/system	directory containing important information about and customize scripts for each fileset
/system/*fileset***/index**	provides information about the fileset, in particular, its version (**update** uses the version to decide if a dependee fileset is already current on the system and need not be reloaded)
/system/*fileset***/update_dest**	names the destination directory under which a fileset was loaded, if other than **/**, for later use by **sam** when clustering a system
/tmp/update.log	log file describing the events that occurred during the update process, including errors, warnings, and notes
/tmp/update.procs	list of unexpected processes running concurrently with **update**; only created if a related warning is reported
/tmp/update.killall	script to kill unexpected processes; only created if a related warning is reported
/tmp/update.cleanup	list of files logged as non-removable, usually due to "text file busy"
/tmp/update.script	file written during interactive invocation for later non-interactive invocation and for readback of default values
/tmp/updist.script	same, but emitted by **updist**
/tmp/update.kernbld	used by cluster nodes to rebuild their kernels at the next reboot
/tmp/customize	script left on the system when some customize scripts are executed after a reboot onto a newly built kernel (runs in **/** and moves itself to **/tmp** when done)
/etc/interface.lib/jam	directory containing JAM user interface information
/etc/update.lib	directory containing files used by **update** and **updist** when run interactively
/etc/update.lib/codeword	saves the last valid *codeword* provided to an invocation of **update** or **updist**
/etc/services	networking services database, file describing networking services, including the netdist service
/etc/checklist	list of volumes that should be mounted
/etc/mnttab	list of volumes currently mounted

SEE ALSO

tar(1), fpkg(1M), rmfn(1M), sam(1M), mount(1M), netdistd(1M), update(4).
HP-UX System Administration and *Installing and Updating HP-UX* manuals.

NAME

uucheck - check the uucp directories and permissions file

SYNOPSIS

/usr/lib/uucp/uucheck [**-v**] [**-x** *debug_level*]

DESCRIPTION

uucheck checks for the presence of the files and directories required by **uucp** (see *uucp*(1)). **uucheck** is executed from the UUCP makefile before the installation occurs. **uucheck** also checks for various obvious errors in the **/usr/lib/uucp/Permissions** file.

Options

uucheck recognizes the following options and command-line arguments:

-v (verbose) Print a detailed explanation of how **uucp** programs will interpret the **Permissions** file.

-x *debug_level*
Debug. *debug_level* is a single digit; the higher the number, the more detail returned.

Note that **uucheck** can only be used by the super-user or **uucp**.

DEPENDENCIES

HP Clustered Environment:

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
/usr/spool/uucp/*
/usr/spool/uucp/LCK*
/usr/spool/uucppublic/*

SEE ALSO

uucp(1), uustat(1), uux(1), uucico(1M), uusched(1M).

UUCP tutorial in *Remote Access User's Guide* .

NAME

uucico - transfer files for the uucp system

SYNOPSIS

/usr/lib/uucp/uucico -r1 -s *system* [**-x** *debug_level*] [**-d** *spool_directory*]

/usr/lib/uucp/uucico [**-x** *debug_level*] [**-d** *spool_directory*]

DESCRIPTION

uucico scans the **/usr/spool/uucp** directories for work files. If such files exist, a connection to a remote system is attempted using the line protocol for the remote system specified in file **/usr/lib/uucp/Systems**. **uucico** then executes all requests for work and logs the results.

Options

uucico recognizes the following options:

-r1 Start **uucico** in the MASTER mode. The default is SLAVE mode.

-s *system* Do work only for the system specified by *system*. If there is no work for *system* on the local spool directory, initiate a connection to *system* to determine if *system* has work for the local system. This option must be used if **-r1** is specified.

-d *spool_directory*
Search directory *spool_directory* instead of the default spool directories (usually **/usr/spool/uucp/***).

-x *debug_level* Use debugging option. *debug_level* is an integer in the range 1 through 9. More debugging information is given for larger values of *debug_level*.

uucico is usually started by a local program such as **cron**, **uucp**, or **uuxqt** (see *cron*(1M), *uucp*(1), and *uuxqt*(1M)). It when debugging should a user initiate **uucico** directly.

When started by a local program, **uucico** is considered the MASTER and attempts a connection to a remote system. If **uucico** is started by a remote system, it is considered to be in SLAVE mode.

For the **uucico** connection to a remote system to be successful, there must be an entry in the **/etc/passwd** file on the remote system of the form:

```
uucp::5:5::/usr/spool/uucppublic:/usr/lib/uucp/uucico
```

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system. See the discussion of the **Permissions** file **MYNAME** parameter in the *Remote Access User's Guide*.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
/usr/spool/uucp/*
/usr/spool/uucp/LCK*
/usr/spool/uucppublic/*

SEE ALSO

uucp(1), uustat(1), uux(1), cron(1M), uusched(1M), uutry(1M).

UUCP tutorial in *Remote Access User's Guide* .

NAME

uuclean - uucp spool directory clean-up

SYNOPSIS

`/usr/lib/uucp/uuclean` [*options*]

DESCRIPTION

`uuclean` scans the spool directories for files with the specified prefix and deletes all those that are older than the specified number of hours.

Options

`uuclean` recognizes the following options:

`-d`*directory* Clean *directory* instead of the spool directory. If *directory* is not a valid spool directory, it cannot contain "work files"; i.e., files whose names start with `C.`. These files have special meaning to `uuclean` pertaining to `uucp` job statistics.

`-p`*pre* Scan for files with *pre* as the file prefix. Up to 10 `-p` arguments can be specified. A `-p` without any *pre* following will cause all files older than the specified time to be deleted.

`-n`*time* Files whose age is more than *time* hours are deleted if the prefix test is satisfied (default time is 72 hours).

`-w`*file* The default action for `uuclean` is to remove files that are older than a specified time (see `-n` option). The `-w` option is used to find files older than *time* hours; however, the files are not deleted. If the argument *file* is present the warning is placed in *file*; otherwise, the warnings go to the standard output.

`-s`*sys* Only files destined for system *sys* are examined. Up to 10 `-s` arguments can be specified.

`-m`*file* The `-m` option sends mail to the owner of the file when it is deleted. If a *file* is specified, an entry is placed in *file*.

This program is typically started by `cron` (see *cron*(1M)).

HP Clusters

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

WARNINGS

`uuclean` works with "old style" `uucp`. See *uucleanup*(1M) for an alternative interface that works with "HoneyDanBer style" `uucp`.

FILES

`/usr/lib/uucp`	directory with commands used by `uuclean` internally
`/usr/spool/uucp/*`	spool directory

SEE ALSO

uucp(1), uux(1), cron(1M), uucleanup(1M).

UUCP tutorial in *Remote Access User's Guide* .

NAME

uucleanup - uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uucleanup [**-C** *time*] [**-D** *time*] [**-W** *time*] [**-X** *time*] [**-m** *string*] [**-o** *time*] [**-s** *system*] [**-x** *debug_level*]

DESCRIPTION

uucleanup scans the spool directories for old files and takes appropriate action to remove them. Depending on the options selected, **uucleanup** performs the following:

- Informs the requestor of send and/or receive requests for systems that cannot be reached.
- Returns mail that cannot be delivered to the sender.
- Removes all other files.

In addition, **uucleanup** warns users of requestors who have been waiting for a given number of days (the default is 1 day). Note that unless *time* is specifically set, the default *time* values for the following options are used.

Options

uucleanup recognizes the following options:

-C*time*	Any **C.** files greater or equal to *time* days old are removed with appropriate information to the requestor. The default *time* is 7 days.
-D*time*	Any **D.** files greater or equal to *time* days old are removed. An attempt is made to deliver mail messages and execute news when appropriate. The default *time* is 7 days.
-W*time*	Any **C.** files equal to *time* cause a message to be mailed to the requestor warning about the delay in contacting the remote. The message includes the *JOBID*, and in the case of mail, the mail message. The administrator can include a message line telling who to call to correct the problem (see the **-m** option). The default *time* is 1 day.
-X*time*	Any **X.** files greater than or equal to *time* days old are removed. The **D.** files are probably not present (if they were, the **X.** could be executed). But, if **D.** files are present, they are taken care of by **D.** processing. The default *time* is 2 days.
-m*string*	This string is included in the warning message generated by the **-W** option. The default string is **See your local administrator to locate the problem.**
-o*time*	Other files whose age is more than *time* days are deleted. The default time is 2 days.
-s*system*	Clean-up the spool directory for *system* only. The default is to clean-up all spool directories.
-x*debug_level*	The debug level is a single digit between 0 and 9. The higher the numbers, the more detailed the debugging information returned.

This program is typically started by the script **uudemon.cleanup**, which should be started by **cron** (see *cron*(1M)).

Clustered Systems

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

FILES

/usr/lib/uucp	directory of commands used by *uucleanup* internally
/usr/spool/uucp/*	spool directory

SEE ALSO

cron(1M), uucp(1), uux(1), uuclean(1M).

UUCP tutorial in *Remote Access User's Guide* .

NAME

uugetty - set terminal type, modes, speed and line discipline

SYNOPSIS

/usr/lib/uucp/uugetty [**-h**] [**-t** *timeout*] [**-r**] *line* [*speed* [*type* [*linedisc*]]]
/usr/lib/uucp/uugetty -c *file*

DESCRIPTION

uugetty sets terminal type, modes, speed and line discipline. It is similar to **getty** , except that **uugetty** supports using the line in both directions (see *getty*(1M)). This allows users to log in, but, if the line is free, **uucico**, **cu**, and **ct** can dial out (see *uucico*(1), *cu*(1), and *ct*(1)). When devices are used with **uucico**, **cu**, and **ct**, lock files are created. Therefore, when the call to **open( )** returns (see *open*(2)) (or the first character is read when the **-r** option is used), the status of the lock files indicates whether the line is used by **uucico**, **cu**, **ct**, or someone trying to log in. See *getty*(1M) for more information.

Note that with the **-r** option, several carriage-return characters might be required before the login message is output. When **uucico** is trying to log in, it can be instructed to enter numerous carriage-return characters with the following login script:

```
\r\d\r\d\r\d\r in:-in: ...
```

where ... represents whatever would normally be used for the login sequence.

An entry for an intelligent modem or direct line that has a **uugetty** on each end must use the **-r** option (this causes *uugetty* to wait to read a character before it enters the login message, thus preventing two instances of **uugetty** from looping). If there is a **uugetty** on one end of a direct line, there must be a **uugetty** on the other end as well.

Clustered Systems

In the HP Clustered Environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

EXAMPLES

The following line is an **/etc/inittab** entry using **uugetty** on an intelligent modem or direct line:

```
30:2:respawn:/usr/lib/uucp/uugetty -r -t 60 tty12 1200
```

WARNINGS

ct does not work when **uugetty** is used with an intelligent modem such as a Penril or a Ventel.

FILES

/etc/gettydefs
/etc/issue
/usr/spool/uucp/LCK*

SEE ALSO

ct(1), cu(1), login(1), uucico(1M), getty(1M), init(1M), ioctl(2), gettydefs(4), inittab(4), tty(7).

UUCP tutorial in *Remote Access User's Guide* .

NAME

uuid_gen - UUID generating program

SYNOPSIS

`/etc/ncs/uuid_gen [-c][-p][-C][-P][-t][-` *version* `]`

DESCRIPTION

uuid_gen generates Universal Unique Identifiers (UUIDs).

Without options, `uuid_gen` generates the character-string representation of a UUID. The `-c` and `-p` options are used when generating templates for Network Interface Definition Language (NIDL) files. The `-C` and `-P` options generate source-code representations of UUIDs, suitable for initializing variables of type `uuid_$t`.

Multiple options can be used in the same command line to generate several representations for the same UUID. To generate the default character-string representation as well as one of the optional representations, use the `-t` option.

Options

`uuid_gen` recognizes the following options:

`-C` Generate the C source-code representation of a UUID.

`-c` Generate a template, including a UUID attribute, for an interface definition in the C syntax of NIDL.

`-Pf1` Generate the Pascal source-code representation of a UUID.

`-pf1` Generate a template, including a UUID attribute, for an interface definition in the Pascal syntax of NIDL.

`-t` Generate the character-string representation of a UUID. This option allows you to request the default output of `uuid_gen` while also requesting optional output forms.

`-version`
Display the version of NCK that this `uuid_gen` belongs to, but do not generate a UUID.

EXAMPLES

Generate the character-string representation of a UUID:

```
/etc/ncs/uuid_gen
34dc23469000.0d.00.00.7c.5f.00.00.00
```

Generate a template for an interface definition in the C syntax of NIDL:

```
$ /etc/ncs/uuid_gen -c
%c
[
uuid(34dc239ec000.0d.00.00.7c.5f.00.00.00),
version(1)
]
interface INTERFACENAME {

}
```

Generate the C source-code representation of a UUID:

```
$ /etc/ncs/uuid_gen -C
= { 0x34dc23af,
    0xf000,
    0x0000,
    0x0d,
    {0x00, 0x00, 0x7c, 0x5f, 0x00, 0x00, 0x00} };
```

Generate both the character-string representation and the C source-code representation of a UUID:

```
$ /etc/ncs/uuid_gen -t -C
450ccaed6000.0d.00.02.18.cb.00.00.00
= { 0x450ccaed,
```

```
            0x6000,
            0x0000,
            0x0d,
            {0x00, 0x02, 0x18, 0xcb, 0x00, 0x00, 0x00} };
```

SEE ALSO

uuid_gen(1M)

Managing NCS Software

NAME

uuls - list spooled uucp transactions grouped by transaction

SYNOPSIS

uuls [**-m**] [*directories* ...]
uuls [**-s**] [**-m**] [*directories* ...]
uuls [**-k**] [**-m**] [*directories* ...]

DESCRIPTION

This command lists the contents of UUCP spool directories (default **/usr/spool/uucp/***) with the files in each directory grouped into three categories:

- Transactions,
- Orphans, and br • Others.

Transactions

Each output line starts with a transaction control filename, and includes the name of each local (same-directory) subfile referenced by the control file (see below). Each is possibly followed by the total size in bytes (**-s** option) or Kbytes (**-k** option) in the transaction (see below). The **-m** (meanings) option replaces the subfile names with nodename, user, and *commandline* information (see below).

Orphans

All subfiles not referenced by any control file.

Others

All other files in the directory (all files not listed under one of the above categories).

Filenames are formatted into columns, so there can be more than one file per line. If a transaction has more subfiles than fit on one line, it is followed by continuation lines which are indented further.

The **-s** (size in bytes) and **-k** (Kbytes) options cause the command to follow each transaction in the **Transactions** section with a total size for all stat-able, sendable files in that transaction. This includes **D.*** files only, not **C.*** or **X.*** files. It does include stat-able files outside the spool directory that are indirectly referenced by **C.*** files. Sizes are either in bytes or rounded to the nearest Kbyte (1024 bytes), respectively. A totals line is also added at the end of the **Transactions** section.

The **-m** (meanings) option causes the command to follow **C.*** and **X.*** files with a *nodename*!*username commandline* line, instead of subfilenames. For **C** files, one line is printed per remote execution (**D*X***) subfile it references. *nodename* is truncated at seven characters, *username* at eight, and *commandline* at however much fits on one line.

If **-m** is given, for each **C** file with no remote execution files, the command instead shows the meaning of the **C** file itself on one or more lines. Each line consists of a username, then **R** (receive) or **S** (send), then the name of the file to be transferred. See below for details.

Filenames are listed in ascending collation order within each section (see Environment Variables below), except that the first section is only sorted by the control filename. Every file in the directory except . and .. appears exactly once in the entire list, unless **-m** is used.

Details

Transaction files are those whose names start with **C.** or **X.**. Subfilenames, which usually start with **D.**, are gleaned from control file lines, at most one per line, from blank-separated fields, as follows:

C.*: R <remotefrom> <localto> <user> -<options>
C.*: S <localfrom> <remoteto> <user> -<options> <subfile> <mode>
X.*: F <subfile>

Lines that do not begin with the appropriate character (**R**, **S**, or **F**) are ignored.

In the **R** (receive) case, <remotefrom> is used to print the **C**-file meaning, and its transaction size is taken as zero (unknown).

In the **S** (send) case, if <subfile> is **D.0**, <localfrom> is a file not in the spool directory, resulting from a typical **uucp** call without the **-C** (copy) option. In this case <localfrom> is used for the transaction size, if stat-able, and to print the **C**-file meaning.

uucp -C and **uux** both set <subfile> to a true (spooled) subfile name.

Orphan files are those whose names start with **D.** and which are not referenced by any control files.

This algorithm extracts from control files the names of all subfiles that should exist in the spool directory when the transaction is not being actively processed. It is not unusual to see "missing subfiles" and "orphans" if you **uuls** a spool directory while **uucico**, **uucp**, **uux**, or **uuxqt** is active.

Meanings information is obtained by reading each **D*X*** subfile referenced by each **C.*** file, and by reading **X*X*** files. *nodename!username* is taken from the last line in the file which is of the form:

U <username> <nodename>

Likewise, *commandline* is taken from the last line of the form:

C <commandline>

If a subfile name is referenced more than once, references after the first show the subfile as missing. If a subfile name appears in a (corrupt) directory more than once, the name is only found once, but then it is listed again under **Orphans**.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

If LC_COLLATE is not specified in the environment or is set to the empty string, the value of LANG is used as a default. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *uuls* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

DEPENDENCIES

Series 300/400

The uucp spool files are found in the **/usr/spool/uucp** directory, instead of in subdirectories of that directory.

AUTHOR

uuls was developed by HP.

SEE ALSO

mail(1), uucp(1), uuto(1), uux(1), uuxqt(1M), stat(2).

UUCP, tutorial in *Remote Access User's Guide* .

DIAGNOSTICS

The program writes an appropriate message to standard error if it has any problems dealing with a specified file (directory), including failure to get heap space. It always returns zero as its exit value.

If a control file is unopenable (wrong permissions or it disappeared while **uuls** was running), its name is preceded by a * and the size of the transaction is zero. If a subfile is missing (filename not found in the directory being listed) or not stat-able (if required for **-s** or **-k**), its name is preceded by a * and it contributes zero bytes to the size of the transaction.

If **-m** is specified and a **D*X*** file is missing or unreadable, its name is given with a * prefixed, as usual.

BUGS

This command uses *chdir*(2) to change to each directory in turn. If more than one is specified, the second through last directories must be absolute (not relative) pathnames, or the *chdir*() may fail.

NAME

uusched - schedule uucp transport files

SYNOPSIS

/usr/lib/uucp/uusched [**-u** *debug_level*] [**-x** *debug_level*]

DESCRIPTION

uusched is the UUCP file transport scheduler. It is usually started by the daemon **uudemon.hour**, which is started by **cron** (see *cron*(1M)) from the following entry in **/usr/spool/cron/crontab**:

```
39 * * * * /bin/su uucp -c */usr/lib/uucp/uudemon.hour > /dev/null*
```

Options

uusched recognizes two options which are provided for debugging purposes only.

-x *debug_level*	Output debugging messages.
-u *debug_level*	Pass as **-x** to **uucico** (see *uucico*(1M)). The *debug_level* is a number between 0 and 9. The higher the number, the more detailed the information returned.

DEPENDENCIES

Clustered Systems

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/spool/uucp/*
/usr/spool/uucp/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), uucico(1M), uusched(1M), uucp(1), uustat(1), uux(1).

UUCP tutorial in *Remote Access User's Guide* .

NAME

uusnap - show snapshot of the UUCP system

SYNOPSIS

uusnap

DESCRIPTION

uusnap displays in tabular format a synopsis of the current UUCP situation. The format of each line is as follows:

site N Cmds N Data N Xqts Message

Where *site* is the name of the site with work, *N* is a count of each of the three possible types of work (command, data, or remote execute), and *Message* is the current status message for that site as found in the **STST** file.

Included in *Message* may be the time left before UUCP can re-try the call, and the count of the number of times that UUCP has tried to reach the site. The process id of **uucico** may also be shown if it is in a TALKING state.

AUTHOR

uusnap was developed by the University of California, Berkeley.

SEE ALSO

uucp(1).

UUCP tutorial in *Remote Access User's Guide* .

NAME

uusnaps - sort and embellish uusnap output

SYNOPSIS

uusnaps

DESCRIPTION

uusnaps runs **uusnap** (see *uusnap*(1M)) and post-processes the output into a more useful form. It sorts output lines in "Pareto-style", showing first those remote systems with the greatest number of **Cmds** files, next **Data** files, and then **Xqts** files.

uusnaps inserts a * after the number of **Xqts** files on those lines where **Data** is not equal to (2 × **Cmds**) + **Xqts**. This may be a sign of missing or orphaned transaction parts. Use **uuls** to check (see *uuls*(1)).

uusnaps adds summary information after all **uusnap** output. The first line is a total of the numbers of **Cmds**, **Data**, and **Xqts** files. The second line contains a grand total number of transaction files, followed by the number of directory bytes this represents. This is an indication of the true size of the directory itself if all empty entries were squeezed out. Finally, if it appears that transaction files might be missing or orphaned, **uusnaps** returns the number of missing or excess files.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

WARNINGS

uusnaps assumes that each directory entry takes 24 bytes.

SEE ALSO

uusnap(1M), uuls(1).

UUCP, tutorial in *Remote Access User's Guide* .

NAME

uusub - monitor uucp network

SYNOPSIS

`/usr/lib/uucp/uusub` [*options*]

DESCRIPTION

`uusub` defines a `uucp` subnetwork and monitors the connection and traffic among the members of the subnetwork.

Options

`uusub` recognizes the following options:

`-a`*sys*	Add *sys* to the subnetwork.
`-d`*sys*	Delete *sys* from the subnetwork.
`-l`	Report the statistics on connections.
`-r`	Report the statistics on traffic amount.
`-f`	Flush the connection statistics.
`-u`*hr*	Gather the traffic statistics over the past *hr* hours.
`-c`*sys*	Exercise the connection to the system *sys*. If *sys* is specified as `all`, exercise the connection to all the systems in the subnetwork.

The connections report is formatted as follows:

sys #call #ok time #dev #login #nack #other

Format interpretation:

sys	remote system name,
#call	number of times the local system tried to call *sys* since the last flush was done,
#ok	number of successful connections,
time	latest successful connect time,
#dev	number of unsuccessful connections because of no available device (e.g., ACU),
#login	number of unsuccessful connections because of login failure,
#nack	number of unsuccessful connections because of no response (e.g. line busy, system down),
#other	number of unsuccessful connections because of other reasons.

Traffic statistics are reported as follows:

sfile sbyte rfile rbyte

Format interpretation:

sfile	number of files sent,
sbyte	number of bytes sent over the period of time indicated in the latest *uusub* command with the `-u`*hr* option,
rfile	number of files received,
rbyte	number of bytes received.

The command:

```
uusub -c all -u 24
```

is typically started by `cron` once a day.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

FILES

`/usr/lib/uucp/L_sub`	connection statistics
`/usr/lib/uucp/R_sub`	traffic statistics
`/usr/spool/uucp/.Log/*`	system log file

SEE ALSO

uucp(1), uustat(1).

UUCP tutorial in *Remote Access User's Guide* .

NAME

uuxqt - execute remote uucp or uux command requests

SYNOPSIS

/usr/lib/uucp/uuxqt [**-s** *system*] [**-x** *debug_level*]

DESCRIPTION

uuxqt executes remote job requests generated by use of the **uux** command (see *uux*(1)). **uux** generates **X.** files and places them in the spool directory, where **uuxqt** searches for them. For each **X.** file, **uuxqt** determines whether the required data files are available and accessible, and if file commands are permitted for the requesting system. The **Permissions** file is used to validate file accessibility and command execute permission. then performs execution of the commands.

Two environment variables are set before the **uuxqt** command is executed: **UU_MACHINE** is the machine that sent the previous job and **UU_USER** is the user who sent the job. These can be used in writing commands that remote systems can execute to provide information, auditing, or restrictions.

uuxqt recognizes the following options:

-s *system* Execute commands on the specified *system*.

-x *debug_level* Produce debugging output on standard output. *debug_level* is a single digit between 0 and 9. The higher the number, the more detailed debugging information returned.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

See the discussion of the **Permissions** file **MYNAME** parameter in the *Remote Access User's Guide*.

FILES

/usr/lib/uucp/Permissions
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/uucp/LCK*

SEE ALSO

uucp(1), uustat(1), uux(1), uucico(1M).

UUCP tutorial in *Remote Access User's Guide*.

NAME

uxgen - generate an HP-UX system

SYNOPSIS

uxgen [**-s**] *infile*

DESCRIPTION

uxgen is used to build an HP-UX system. The user supplies a set of instructions in *infile* that selects optional parts of the kernel (such as I/O drivers, pseudo-drivers, subsystems, file systems) and specifies values for system parameters such as the location of the swap area.

The files output by **uxgen** are placed in the directory **../***infile*. This directory is created if it does not exist. Four files (**conf.c**, **config.h**, **libs_file**, **Makefile**) are created by **uxgen**. The file **libs_file** contains a macro recognized by **make** specifying the libraries needed to produce a kernel that includes all the subsystems and file systems specified in *infile* (see *make*(1)). In addition to *infile*, the file named **Makefile** must exist in the current directory. **Makefile** is supplied with the system and contains targets for compiling **conf.c** and linking the kernel (**hp-ux**) using the **make** macro defined in **libs_file**. The concatenation of **libs_file** and **Makefile** creates the file **Makefile** in the **../***infile* directory.

After creating **Makefile**, **uxgen** changes the current directory to **../***infile* and executes the **make** command. However, if the **-s** option is specified, **make** is not executed. **make** compiles **conf.c** and links the kernel (**hp-ux**) with the appropriate kernel libraries. File **hp-ux** can then be booted. See the *HP-UX System Administrator Manuals* for information on how to include or remove a subsystem or file system, and how to boot the system.

Many header files are needed to compile **conf.c**. Also, archive library files containing the kernel objects are needed to link the kernel. These files are supplied with the system and are contained in the directories found under **/etc/conf**. The directories in **/etc/conf** can be moved to any location in the file system. However, all the directories must exist before the kernel can be built. By convention, *infile* is placed in the directory named **gen** (usually, **/etc/conf/gen**).

Kernel Building Procedure

To build a new kernel follow these steps, using the commands indicated:

1. Change to the directory containing *infile* (usually **/etc/conf/gen**).
   Command: **cd /etc/conf/gen**
2. Edit or create the input file **/etc/conf/gen/***infile*. On a newly installed system, *infile* is usually named **S800**.
3. Use the command **uxgen** *infile* to build the new kernel file named **../***infile***/hp-ux**.
4. *Copy* the old kernel to a backup file: **cp /hpux /SYSBCKUP**. This ensures that a good kernel is still available to boot from if a problem occurs during the copy operation.
5. Overwrite the old kernel file with the new one.
   Command: **mv ../***infile***/hp-ux /hp-ux**
6. Reboot the system (see *shutdown*(1M)).

Statements used in *infile* form a simple C-like language. **uxgen** first passes *infile* through the **cpp** C preprocessor to allow features such as comments, macros, file inclusion and conditional statements (see *cpp*(1)) before reading the file.

Generally, the first statement in *infile* is **#include /etc/master**. This causes the statements in **/etc/master** to be read prior to the remaining statements in *infile*. File **/etc/master** contains "subsystem", "file system", "driver", "pseudo-driver", and "tunable definition" statements which describe kernel software subsystems, file systems, I/O drivers, pseudo-drivers, tunable parameters, and major number assignments for software supplied by HP. Only users who write kernel software need to understand these statements. They are described in the System Administrator manuals supplied with your system.

It is beyond the scope of this manual entry to give a complete description of the statements that can be used in *infile*. A sketchy syntax description for most statements is given below where:

module_path A module path is a string of names separated by periods, '.'. Each name identifies a driver controlling a hardware component on the path to a device.

hw_path A hardware path specifies the addresses of the hardware components leading to a device. It consists of a string of numbers, each suffixed by a / character, followed by a string of numbers separated by period (.) characters. Hardware components suffixed by / characters indicate bus converters, and may not be necessary on your machine. Hardware components suffixed by . characters indicate the addresses of the remaining hardware components on the path to the device.

disk_spec *module_path* **at** *hw_path* [**minor** *integer*] [**section** *integer*]

See the *HP-UX System Administrator Tasks* manual for more information.

infile **Statements**

console on default [**minor** *integer*];
console on *module_path* **at** *hw_path* [**minor** *integer*];
Specify the system console. The default is the console device used when booting the system (i.e., Page Zero).

dumps on default [**minor** *integer*] [**section** *integer*];
dumps on *disk_spec*;
Specify location used for writing operating system image after operating system detects a fatal error (panic). Dump device defaults to primary swap.

include *identifier*;
Include a pseudo-driver, subsystem, file system, or driver in the kernel.

io {
identifier integer **address** *integer*; ... *identifier* **address** *integer* **{**
identifier **address** *integer* ;
} ...
}
Specify the number of I/O devices and how they are connected. Used to override default mappings from driver to hardware component or to include drivers that do not support autoconfiguration.

remove *identifier*;
Remove a pseudo-driver, subsystem, file system or driver that was previously included with an include statement.

root on default [**minor** *integer*] [**section** *integer*];
root on *disk_spec* [**mirrored on** *disk_spec*];
Specify the root of the file system (/). The default root device is taken from the **hpuxboot** command line. The root of the file system can be mirrored by explicitly specifying both the root device and the mirror device (see *mirror*(1M)).

swap on default [**minor** *integer*] [**section** *integer*] [*disk_spec* ...];
swap on *disk_spec* [**mirrored on** *disk_spec*] [*disk_spec* ...];
Specify on which disk and section each swap area is located. Multiple swap areas can be defined. Only the first swap area listed (i.e., primary swap) can be mirrored (see *mirror*(1M)). The default for primary swap is section 1 of the root device.

tunable-ID integer;
tunable-ID **"***anychars***"** ;
Assign a value to a tunable parameter. *tunable-ID* can be any identifier listed below under Tunable Parameters. All HP tunable parameters are defined and assigned a default value using a "tunable definition" statement in file **/etc/master**.

Tunable Parameters

acctresume The percentage of file system space that must be free to reactivate process accounting after it is suspended due to insufficient free space (see **acctsuspend**).

acctsuspend The percentage of file system space that must be free to allow process accounting (see **acctresume**).

bootspinlocks The number of kernel spinlocks available for allocation during system boot. If the kernel panics during system boot with the message **alloc_spinlock: not enough boot spinlocks**, this parameter should be increased.

bufpages The number of memory pages allocated to the file-system buffer cache. Each page is **NBPG** bytes long (see **<sys/param.h>**). If **bufpages** is zero, two pages are allocated for each buffer header specified by **nbuf**, provided **nbuf** is non-zero. If both **nbuf** and **bufpages** are zero, ten percent of available memory is allocated by the kernel.

check_alive_period
The period between checking for alive messages from an HP cluster cnode (specified in seconds).

dmmax
dmmin
dmshm
dmtext The values of these parameters are used as described below under Swap Space Parameter Interaction.

dskless_cbufs Obsolete. This parameter is no longer used.

dskless_fsbufs
The number of pages allocated to the fsbuf (file system buffer) pool. Each page is **NBPG** bytes long (see **<sys/param.h>**).

dskless_mbufs Obsolete. This parameter is no longer used.

dskless_node A flag that identifies a diskless client node or cluster server. A value of **1** identifies a client node. A value of **0** identifies the cluster server.

dst A flag that specifies whether daylight saving time should be used. A value of **0** means not to use daylight saving time. A value of **1** indicates that U.S.A. daylight saving time should be used.

maxdsiz The maximum size (in pages) of a process's data segment.

maxfiles The maximum number of files a given process can have open simultaneously. This limit can be changed using *setrlimit*(2).

maxfiles_lim A limit on the number of open files a non-super-user process can have. Non-super-user processes cannot increase their open file limit beyond **maxfiles_lim**.

maxssiz The maximum size (in pages) of a process's stack.

maxswapchunks The maximum number of dmmax * 1 kbyte blocks of swap space allocated.

maxtsiz The maximum size (in pages) of a process's shared text segment.

maxuprc The maximum number of processes a user may have.

maxusers The maximum number of expected users. Assigning a value to this parameter causes the macro MAXUSERS to be defined (for example, **#define MAXUSERS 8**). MAXUSERS is used to determine other tunable parameters (for example, **nproc "(20 + 8 * MAXUSERS)";**).

minswapchunks The minimum number of **dmmax** × 1K-byte blocks of swap space allocated.

msgmap The number of message map entries.

msgmax The maximum number of bytes in a message.

msgmnb The total number of bytes allowed for all messages queued on a message queue.

msgmni The number of message queue identifiers.

msgseg The number of units (each **msgssz** bytes long) available for messages.

msgssz The size (in bytes) of each unit of memory used for messages (see **msgseg**).

msgtql The number of message headers.

nbuf The number of file-system buffer cache buffer headers. If both **nbuf** and **bufpages** are set to 0, the kernel allocates ten percent of available memory to buffer space. If only **nbuf** is 0, it will be computed from **bufpages**, assuming 4096 bytes per buffer. If both variables are non-zero, the kernel attempts to adhere to both requests, but if necessary, **nbuf** is changed to correspond to **bufpages**.

ncallout The number of timeouts that can be pending simultaneously.

netisr_priority
If a networking subsystem is configured, **netisr_priority** specifies the realtime priority by which the process **/etc/netisr** is scheduled, and by which all networking runs. Otherwise, **netisr_priority** is ignored.

nfile The maximum number of open files.

nflocks The maximum number of file locks.

ngcsp The number of general cluster server processes (for diskless server).

ninode The maximum number of open in-core inodes.

nproc The maximum number of processes that can exist simultaneously.

npty The number of ptys (pseudo-terminals).

nstlbe The number of Software TLB entries requested.

nswapdev The maximum number of devices that can be enabled for swapping.

nswapfs The number of file systems available for swapping.

ntext The maximum number of active shared text descriptors.

num_cnodes The limiter for diskless system resource allocation. It is used as an indicator of the number of diskless cnodes that a server can reasonably expect to serve simultaneously. Assigning a value to this parameter causes the macro **NUM_CNODES** to be defined (for example, **#define NUM_CNODES 5**). **NUM_CNODES** is used to determine other tunable parameters (for example, **ngcsp "8 * NUM_CNODES";**). **pfail_enabled** Specifies whether recovery from system power loss is enabled. A value of **1** enables this feature. A value of **0** disables it.

retry_alive_period
The period to continue checking for alive messages from an HP cluster cnode (specified in seconds).

scroll_lines The number of lines of emulated terminal memory, both on-screen and off-screen, for each ITE (Internal Terminal Emulator) port.

selftest_period
The period between execution of kernel self-test (specified in seconds).

semaem The maximum value by which a semaphore can be adjusted due to the death of a process.

semmap The number of semaphore map entries.

semmni The number of semaphore identifiers.

semmns The maximum number of semaphores.

semmnu The maximum number of processes that can have pending "semaphore undo" requests on a semaphore.

semume The maximum number of semaphores on which a process can have a pending "semaphore undo" request.

semvmx The maximum value of a semaphore.

server_node A flag used to size an array for the root server's inbound requests. A value of **1** causes the serving_array[ ] and **ninode** to be sized for a server node. A value of **0** causes

the serving_array[ ] and **ninode** to be sized for a cluster cnode.

serving_array_size
The size of the cluster's serving array. The serving array is an array of kernel structures that holds information related to inbound requests.

shmmax The maximum number of bytes in a shared memory segment.

shmmni The maximum number of shared memory segments.

shmseg The maximum number of shared memory segments that can be attached simultaneously to a process.

timeslice The number of 10-millisecond intervals used for round-robin scheduling. A value of **-1** disables round-robin scheduling.

timezone The minutes west of Greenwich.

unlockable_mem
The number of bytes of memory that cannot be locked.

using_array_size
The size of the diskless cnode's using array. The using array is an array of kernel structures that holds information related to outbound requests.

Swap Space Parameter Interaction

If you change **maxdsiz**, **maxssiz**, **maxtsiz**, or **shmmax**, you must also change **dmmin**, **dmmax**, **dmtext**, and **dmshmem**. All of these swap space system parameters interact, and a wrong value might make your virtual memory system unworkable.

Appendix A of the *HP-UX System Administrator Tasks* manual contains a table listing the values these parameters must have for given values of **maxdsiz**, **maxssiz**, **maxtsiz**, or **shmmax**.

EXAMPLES

A typical *infile* resembles the following:

```
include         mirror;
include         nfs;
include         nsdiag0;

include         cio_ca0;
include         hpib0;
include         hpib1;
include         disc1;
include         hpfl0;
include         disc2;
include         mux0;
include         tape0;
include         instr0;
include         gpio0;

console on      default;
root on         cio_ca0.hpfl0.disc2 at 8.3.0 section 0
mirrored on     cio_ca0.hpfl0.disc2 at 8.3.1 section 0;
dumps on        default;
swap on         default
                cio_ca0.hpib0.disc1 at 8.0.0 section 1
                cio_ca0.hpib0.disc1 at 8.0.1 section 1;

dskless_node    1;
maxuprc         30;
maxusers        32;
nproc           "(20 + 8 * MAXUSERS)";
server_node     0;
timezone        420;
io {
        cio_ca0 address 8 {
```

```
                    hpib0 address 2 {
                        instr0 address 7;
                    }
                    gpio0 address 5;
                }
        }
```

AUTHOR

`uxgen` was developed by HP.

FILES

`/etc/master`

SEE ALSO

make(1), insf(1m), cpp(1m), lssf(1m), mirror(1m), mksf(1m), shutdown(1m), privilege(5).

NAME

vgcfgbackup - create or update volume group configuration backup file

SYNOPSIS

vgcfgbackup [**-u**] [**-f** *vg_conf_path*] *vol_group_name*

DESCRIPTION

vgcfgbackup saves LVM configuration for volume group *vol_group_name* in *vg_conf_path* if provided, otherwise in **/etc/lvmconf/***vol_group_name***.conf** by default.

vol_group_name is the name of the directory containing volume group special files and is extracted from *vol_group_name* (e.g., if *vol_group_name* is specified as **/dev/vg00**, *vol_group_name* extracted is **vg00**).

The **-u** option updates the configuration file with the latest LVM configuration. Note that all physical volumes for *vol_group_name* should be on-line to create a new configuration file. This condition is not necessary for update (**-u**), in which case only those physical volumes added since the last backup was taken need to be on-line.

Options

vgcfgbackup recognizes the following options and arguments:

-f *vg_conf_path*
Saves configuration in file specified by *vg_conf_path*.

-u Updates configuration file with latest LVM configuration.

Note: It is recommended that the back-up file be created in the root file system (as is the case with the default pathname) to facilitate easy volume group recovery during maintenance mode, such as after a system crash.

RETURN VALUE

vgcfgbackup exits with one of the following values:

0 LVM configuration was saved successfully.

>0 **vgcfgbackup** aborted because errors occurred when accessing information from volume group.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **vgcfgbackup** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Back up LVM configuration information for volume group **/dev/vg00** in the default file **/etc/lvmconf/vg00.conf**:

```
vgcfgbackup /dev/vg00
```

Update LVM configuration information corresponding to volume group **/dev/vg00** in the default file **/etc/lvmconf/vg00.conf**:

```
vgcfgbackup -u /dev/vg00
```

Back up LVM configuration information for volume group **/dev/vg00** in the file **/tmp/vg00.backup**:

```
vgcfgbackup -f /tmp/vg00.backup vg00
```

WARNINGS

All Physical Volumes must be on-line when creating a new configuration file.

AUTHOR

vgcfgbackup was developed by HP.

SEE ALSO

vgcfgrestore(1M).

NAME

vgcfgrestore - restore volume group configuration from a configuration file

SYNOPSIS

vgcfgrestore {**-n** *vol_group_name* | **-f** *vg_conf_path* } **-l**
vgcfgrestore {**-n** *vol_group_name* | **-f** *vg_conf_path* } [**-o** *old_pv_path*] *pv_path*

DESCRIPTION

vgcfgrestore restores LVM configuration from configuration file **/etc/lvmconf/***vol_group_name***.conf** or **vg_conf_path** to Physical Volume *pv_path* depending upon options chosen.

vol_group_name is the directory name containing the volume group's special files, and is extracted from *vol_group_name* (e.g., if *vol_group_name* specified is **/dev/vg00**, *vol_group_name* extracted is **vg00**).

Use the **-o** option to restore configuration saved for *old_pv_path* to the disk specified by *pv_path* (e.g., when a Physical Volume's name has changed since the backup was taken using *vgcfgbackup*(1M)).

The **-l** option lists the configuration information saved in the configuration file.

If **-o** is not specified, *pv_path* must belong to the volume group corresponding to the configuration file. Otherwise *old_pv_path* must belong to the volume group when **vgcfgbackup** was executed.

Options

vgcfgrestore recognizes the following options and accompanying arguments:

-n *vol_group_name*
: Identifies the configuration file (**/etc/lvmconf/***vol_group_name***.conf**) that contains the configuration information. See above for description of *vol_group_name*. **-n** cannot be used with the **-f** option.

-f *vg_conf_path*
: Identifies the configuration file (*vg_conf_path*) that contains the configuration information. The **-f** option cannot be used with the **-n** option.

-l
: Lists configuration information saved in the configuration file.

-o *old_pv_path*
: Used to restore configuration information saved for physical volume *old_pv_path* to *pv_path*.

RETURN VALUE

vgcfgrestore exits with one of the following values:

0 LVM configuration was restored successfully.

>0 **vgcfgrestore** aborted because errors occurred during the restore operation.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **vgcfgrestore** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Restore the LVM configuration information for the physical volume **/dev/rdsk/c7d0s2** that was saved in the default file **/etc/lvmconf/vg00.conf**:

```
vgcfgrestore -n /dev/vg00 /dev/rdsk/c7d0s2
```

Restore LVM configuration information to physical volume **/dev/rdsk/c4d0s2** using configuration file **/tmp/vg00.backup**:

```
vgcfgrestore -f /tmp/vg00.backup /dev/rdsk/c4d0s2
```

List back up information saved in configuration file **/etc/lvmconf/vg01.conf**:

```
vgcfgrestore -n /dev/vg01 -l
```

Restore LVM configuration information stored for `/dev/rdsk/c7d0s2` in configuration file `/etc/lvmconf/vg01.conf` to physical volume `/dev/rdsk/c6d0s2`:

```
vgcfgrestore -n /dev/vg01 -o /dev/rdsk/c7d0s2 /dev/rdsk/c6d0s2
```

WARNINGS

Volume group should preferably be made non-available before executing `vgcfgrestore` by using `vgchange -a n` *vol_group_name*.

AUTHOR

`vgcfgrestore` was developed by HP.

SEE ALSO

vgcfgbackup(1M).

NAME

vgchange - set volume group availability

SYNOPSIS

/etc/vgchange -a *availability* [**-q** *quorum*] [**-l**] [**-p**] [**-s**] *volume_group_name*

DESCRIPTION

vgchange activates or deactivates one or more volume groups as specified by the **-a** option; namely **y** or **n**:

vgchange -a n
Deactivate *volume_group_name* and its associated logical volumes. Close the logical volumes prior to executing **vgchange -a n**. For example, if the logical volume contains a file system, the file system must be unmounted.

vgchange -a y
Activate volume group *volume_group_name* and all associated physical and logical volumes. When a volume group is activated, LVM performs the necessary mirror consistency recovery for each logical volume in the volume group based on the state of the Mirror Write Cache and Mirror Consistency Recovery (see Consistency Recovery section of **lvdisplay**(1M)).

MWC Mirror consistency is recovered by using the Mirror Write Cache and Mirror Consistency Record. This mode implies that the Mirror Write Cache is on.

NOMWC Mirror consistency is recovered by searching all logical extents and copying data from a **non-stale** copy to the other mirror copies. This mode implies that the Mirror Write Cache is off.

NONE No mirror consistency recovery during volume group activation on this logical volume. This mode implies that the Mirror Write Cache is off.

The next step following mirror consistency recovery is mirror synchronization which refreshes **stale** mirror copies by copying data from a **non-stale** copy. If the **-s** option is specifed on the command line, mirror synchronization does not occur. However, for those logical volumes that have Mirror Write Cache turned off, mirror synchronization is done by a daemon (**/etc/nomwcsyncd**) independent of whether the **-s** option appears on the command line.

If **vgchange** cannot access a physical volume, it lists the volume's status as missing. If too many physical volumes in the volume group are missing, **vgchange** complains that the group does not have a quorum and cannot be activated. The **-q n** option is used to activate the volume group when disk quorum is not maintained because too many disks were lost. The **-p** option is used to activate the volume group only if all of the physical volumes belonging to the volume group are available. If the **-l** option is set, later attempts to open the logical volumes fail. To make an open of these logical volumes succeed, execute **lvchange -a y**.

vgchange -a y on a currently active Volume Group attempts to include any Physical Volumes that were previously listed as missing. This is useful if a Physical Volume has come back online. However, no automatic synchronization of any mirrored Logical Volumes is done. If synchronization is required, execute the **vgsync** command (see *vgsync*(1M)).

Options

vgchange recognizes the following options and arguments:

-a *availability*
Set volume group availability. *availability* can have either of the following values:

y Make a volume group available.

n Make a volume group temporarily unavailable.

-q *quorum*
Set activation quorum requirement for the volume group. *quorum* can have either of the following values:

`y` (default) Enforce quorum requirement.

`n` Ignore quorum requirement.

`-l`
Disable open of logical volumes that belong to the volume group.

`-p`
Activate volume group only if all of the physical volumes that belong to it are available.

`-s`
Disable synchronization of **stale** physical extents within the volume group specified by *volume_group_name*. This option only appropriate when used with the `-a y` option.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **vgchange** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Activate a volume group:

```
vgchange -a y /dev/vg03
```

Deactivate a volume group:

```
vgchange -a n /dev/vg03
```

Activate a volume group without synchronizing extents that are not current on logical volumes that have Mirror Write Cache turned on:

```
vgchange -a y -s /dev/vg03
```

FILES

`/etc/nomwcsyncd` Mirror sychronization daemon for logical volumes that have Mirror Write Cache turned off.

SEE ALSO

vgcreate(1M), vgextend(1M), vgreduce(1M), vgdisplay(1M), umount(1M).

NAME

vgcreate - create a volume group

SYNOPSIS

/etc/vgcreate [**-x** *extensibility*] [**-e** *max_physical_extents*] [**-l** *max_logical_vols*] [**-p** *max_physical_vols*] [**-s** *physical_extent_size*] [**-g** *physical_vol_group_name*] *volume_group_name* *physical_volume_path* ...

DESCRIPTION

vgcreate creates a new volume group. *volume_group_name* is a symbolic name for the volume group and must be used in all references to it. *volume_group_name* is the path to a directory entry under **/dev** which must contain a character special file named **group**. Except for the **group** entry, the directory *volume_group_name* should not contain any other entries.

vgcreate leaves the volume group in an active state.

Before assigning a physical volume to a volume group, the physical volume has to be created using the **pvcreate** command (see *pvcreate*(1M)).

If **vgcreate** fails to install the first specified physical volume into the volume group, the volume group is not created. If, for any reason, one of the remaining specified physical volumes cannot be installed into the volume group, an error message is printed, but the installation continues until the end of the list of physical volumes.

Options and Arguments

-x *extensibility*

Set the allocation permission for adding physical extents on the physical volumes specified by the *physical_volume_path* parameter. *extensibility* can have either of the following values:

y (default) Allow allocation of additional physical extents on the physical volume.

n Prohibit allocation of additional physical extents on the physical volume. Logical volumes residing on the physical volume can still be accessed after the volume group has been activated by the **vgchange -a y** command.

-e *max_physical_extents*

Set maximum number of physical extents that can be allocated from any of the physical volumes in the volume group. Default value for *max_physical_extents* is **1016**. However, if the size of any physical volume exceeds **1016** times the *physical_extent_size*, the default value for *max_physical_extents* is adjusted to match the physical volume size. The maximum number of physical extents can be a value ranging from 1 through 65 535.

-l *max_logical_vols*

Set maximum number of logical volumes that the volume group is allowed to contain. Default value for *max_logical_vols* is **255**. The maximum number of logical volumes can be a value ranging from 1 through 255.

-p *max_physical_vols*

Set maximum number of physical volumes that the volume group is allowed to contain. Default value for *max_physical_vols* is **16**. The maximum number of physical volumes can be a value ranging from 1 through 255.

-s *physical_extent_size*

Sets the number of megabytes in each physical extent, where *physical_extent_size* is expressed in units of Mbytes from 1 through 256. *physical_extent_size* must be equal to a power of 2 (1, 2, 4, 8, etc.). Default value for *physical_extent_size* is **4** Mbytes.

-g *physical_vol_group_name*

Create a new physical volume group with the name *physical_vol_group_name*. All physical volumes specified in the *physical_volume_path* parameter become a member of the newly created physical volume group.

The physical volume group information is stored in an ASCII file **/etc/lvmpvg**. The file can be edited to

create a physical volume group instead of using the **vgcreate** command. However, ensure that the physical volumes to be used have already been installed in the volume group prior to creating the physical volume group.

The physical volume group name must be unique within a volume group although identical physical volume group names can appear in different volume groups (see *lvmpvg*(4) for format details).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **vgcreate** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Create a volume group named */dev/vg00* containing two physical volumes with extent size set to 2 Mbytes. If directory **/dev/vg00** exists with the character special file **group**, the volume group is created:

```
vgcreate -s 2 /dev/vg00 /dev/dsk/c1d0s2 /dev/dskc2d0s2
```

Create a volume group named **/dev/vg01** that can contain a maximum of three logical volumes, with extent size set to 8 Mbytes:

```
vgcreate -l 3 -s 8 /dev/vg01 /dev/dsk/c4d0s2
```

Create a volume group named **/dev/vg00** and a physical volume group named **PVG0** with two physical volumes:

```
vgcreate -g PVG0 /dev/vg00 /dev/dsk/c1d0s2 /dev/dsk/c2d0s2
```

REMARKS

It is not possible to create a volume group that contains both HP-IB devices and devices using another type of interface.

SEE ALSO

pvcreate(1M), vgchange(1M), vgdisplay(1M), vgextend(1M), vgreduce(1M).

NAME

vgdisplay - display information about volume groups

SYNOPSIS

/etc/vgdisplay [**-v**] [*volume_group_name ...*]

DESCRIPTION

vgdisplay displays information about volume groups. If *volume_group_name* is specified, **vgdisplay** displays information for that volume group only. If no *volume_group_name* is specified, **vgdisplay** displays names and corresponding information for all defined volume groups.

-v Command-Line Option not Specified

If the **-v** (verbose) option does not appear in the command line, only the following information is displayed:

VG Name: Name of the volume group.

VG Status: State of the volume group:

- **on** Volume group previously activated by **vgchange -a y** command.
- **off** Volume group previously deactivated by **vgchange -a n** command.

Max LV:
Maximum number of logical volumes allowed in the volume group.

Cur LV:
Current number of logical volumes in the volume group.

Open LV:
Number of logical volumes currently open in the volume group.

Max PV:
Maximum number of physical volumes allowed in the volume group.

Cur PV:
Current number of physical volumes in the volume group.

Active PV:
Number of physical volumes that are currently active.

PE Size:
Size of each physical extent.

Max PE per PV:
Maximum number (limit) of physical extents that can be allocated from any of the physical volumes in the volume group.

Total PE:
Total number of physical extents within the volume group: the sum of the number of physical extents belonging to each available physical volume in the volume group.

Alloc PE:
Number of physical extents currently allocated to logical volumes.

Free PE:
Number of physical extents not allocated.

VGDA:
Number of volume group descriptor areas within the volume group.

Total PVG:
Total number of physical volume groups within the volume group.

-v Command-Line Option Specified

If the **-v** command-line option is specified, **vgdisplay** lists additional information for each logical volume, for each physical volume, and for each physical volume group in the volume group:

Logical volumes:
Lists information about logical volumes belonging to *volume_group_name*:

LV Name: Name of logical volume in the volume group.

LV Status: State of the logical volume:

`available`	`available/stale`
`unavailable`	`available/syncd`

`available/stale`
Logical volume available but contains physical extents that are not current.

`available/syncd`
Logical volume available and synchronized.

`available` Logical volume available; stale/syncd state cannot be confidently determined because logical volumes have both the Mirror Write Cache and Mirror Consistency Recovery turned off.

`unavailable`
Logical volume is not available for use.

`LV Size`:
Size of the logical volume.

`Total LE`:
Number of logical extents in the logical volume.

`Used PE`:
Number of physical extents used by the logical volume.

`Used PV`:
Number of physical volumes used by the logical volume.

`Physical volumes`:
Lists information about physical volumes belonging to *volume_group_name*:

`PV Name`: Name of physical volume in the group.

`PV status`: State of the physical volume.

`Total PE`: Total number of physical extents on the physical volume.

`Free PE`: Number of free physical extents on the physical volume.

`Physical Volume Group`:
Lists information about physical volume groups belonging to *volume_group_name*:

`PVG Name`:
Name of physical volume group in the volume group.

`PV Name`:
Name of physical volume in the physical volume group.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **`vgdisplay`** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Display information about all the volume groups within the system:

```
vgdisplay
```

Display all of the information about one volume group, including the characteristics and status of both the logical and physical extents of the volume group:

```
vgdisplay -v /dev/vg02
```

SEE ALSO
lvdisplay(1M), pvdisplay(1M), vgchange(1M), vgcreate(1M).

NAME

vgexport - export a Volume Group and its associated Logical Volumes

SYNOPSIS

/etc/vgexport [**-p**][**-v**][**-m** *mapfile*] *volume_group_name*

DESCRIPTION

vgexport removes a Volume Group from the system without modifying the Logical Volume information as found on the Physical Volumes.

The *volume_group_name* is removed from the **/etc/lvmtab** file, and the associated device files including the *volume_group_name* directory and *group* file are removed from the system.

The Volume Group information and data is untouched on the Physical Volume. These disks can be imported to other system by using the **vgimport** command (see *vgimport*(1M)).

Options

vgexport recognizes the following options:

-p Preview actions taken but do not update file **/etc/lvmtab** or remove the devices file. This option is best used in conjunction with the **-v** option.

-v Print verbose messages including the names of the Physical Volumes associated with this Volume Group.

-m*mapfile*
Specify the name of the file to which Logical Volume names and numbers are to be written. If this option is not specified, no Logical Volume names are saved. This file can be used as input to **vgimport** (see *vgimport*(1M)).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **vgexport** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Export the Volume Group **/dev/vg01** into mapfile **vg01.mapfile**:

```
vgexport -m vg01.mapfile /dev/vg01
```

SEE ALSO

vgimport(1M), vgscan(1M).

NAME

vgextend - extend a volume group by adding physical volumes to it

SYNOPSIS

/etc/vgextend [**-x** *extensibility*] [**-g** *physical_vol_group_name*] *volume_group_name* *physical_volume_path* ...

DESCRIPTION

vgextend assigns physical volumes to *volume_group_name*. The volume group must be active.

Volume groups are extended by adding one or more physical volumes specified by *physical_volume_path*.

After the physical volumes have been successfully added to the volume group, they can be used.

Before assigning a physical volume to a volume group, create the physical volume by use of the **pvcreate** command (see *pvcreate*(1M)).

If, for any reason, one of the remaining specified physical volumes cannot be installed into the volume group, an error message is printed. However, the installation continues to the end of the list of physical volumes.

Options

vgextend recognizes the following command-line options and arguments:

-x *extensibility* Set allocation permission for additional physical extents on the physical volume specified by *physical_volume_path*. *extensibility* can have either of the following values:

- **y** Allow allocation of additional physical extents on the physical volume.
- **n** Prohibit allocation of additional physical extents on the physical volume. logical volumes residing on the physical volume can still be accessed.

-g *physical_vol_group_name*
This option is used to extend an existing physical volume group while the volume group is being extended by adding all the physical volumes in the *physical_volume_path* parameter to the physical volume group specified by *physical_vol_group_name*.

If the specified physical volume group does not exist, it is created, thus providing a means for creating new physical volume groups after the volume group has been created. Another way to extend or add a physical volume group is to edit the **/etc/lvmpvg** file as is described in the **vgcreate** command (see *lvmpvg*(4) for format details).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **vgextend** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Add physical volumes **/dev/dsk/c1d0s2** and **/dev/dsk/c2d0s2** to volume group **/dev/vg03**:

```
vgextend /dev/vg03 /dev/dsk/c1d0s2 /dev/dsk/c2d0s2
```

Extend physical volume group **PVG0** while adding physical volumes **/dev/dsk/c3d0s2** and **/dev/dsk/c4d0s2** to volume group **/dev/vg03**:

```
vgextend -g PVG0 /dev/vg03 /dev/dsk/c3d0s2 /dev/dsk/c4d0s2
```

REMARKS

It is not possible to extend a volume group such that it contains both HP-IB devices and devices using another type of interface.

SEE ALSO

pvchange(1M), pvcreate(1M), vgchange(1M), vgcreate(1M), vgdisplay(1M).

NAME

vgimport - import a Volume Group onto the system

SYNOPSIS

/etc/vgimport [**-p**] [**-v**] [**-m** *mapfile*] *volume_group_name physical_volume_path* ...

DESCRIPTION

vgimport adds the specified Volume Group to the system. The specified Physical Volumes are scanned to obtain the Volume Group information and Logical Volume information. This command works much like **vgcreate** by requiring that the Volume Group device directory and **group** special file be created before the command is executed (see *vgcreate*(1M)). The *volume_group_name* is added from the **/etc/lvmtab** file, and the associated logical volume device files are added to the system.

vgimport assumes that the Volume Group information has already been created on the Physical Volumes. This command is useful in conjunction with the **vgexport** command (see *vgexport*(1M)), to move volume groups from one system to another.

vgimport creates Logical Volume devices files under the **volume_group_name** directory using the naming convention given in the **mapfile** or using the default naming convention used by the **lvcreate** command (see *lvcreate*(1M)).

vgimport does not activate the imported Volume Group due to the many possible options at Volume Group activation time. To activate the Volume Group once it has been successfully imported, use the **vgchange** command (see *vgchange*(1M)).

Options

vgimport recognizes the following options:

-p Preview actions taken but do not update file **/etc/lvmtab** or add the devices file. This option is best used in conjunction with the **-v** option.

-v Print verbose messages including names of the Logical Volumes.

-m*mapfile*

Specify the name of the file from which Logical Volume names and numbers are to be read. If this option is not specified, Logical Volume names are created using the default naming convention **lvol***nn* where *nn* is the Logical Volume minor number.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **vgimport** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Import the Volume Group **/dev/vg01** that is located on Physical Disks **/dev/dsk/c1d0s2** and **/dev/dsk/c3d0s2**:

```
vgimport -v /dev/vg01 /dev/dsk/c1d0s2 /dev/dsk/c3d0s2
```

Activate the volume group following a successful import:

```
vgchange -a y vg01
```

SEE ALSO

vgexport(1M), vgscan(1M).

NAME

vgreduce - reduce a volume group by removing one or more physical volumes

SYNOPSIS

/etc/vgreduce *volume_group_name physical_volume_path ...*

DESCRIPTION

vgreduce removes the physical volume or volumes specified by *physical_volume_path* from *volume_group_name*.

All but one physical volume can be removed. The last physical volume must remain in the volume group so that the logical volume driver can continue to operate. The last physical volume in the volume group is removed by the **vgremove** command (see *vgremove*(1M)).

All logical volumes residing on the physical volume or volumes represented by *physical_volume_path* must be removed by executing **lvremove** before executing **vgreduce** (see *lvremove*(1M)).

Any physical volumes appearing in *physical_volume_path* that is also a member of a physical volume group (as defined in **/etc/lvmpvg**) is also be removed from that physical volume group. If the physical volume happens to be the last one in the physical volume group, the physcial volume group is also removed from the volume group.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **vgreduce** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Remove physical volume **/dev/dsk/c1d0s2** from volume group **/dev/vg01**:

```
vgreduce /dev/vg01 /dev/dsk/c1d0s2
```

SEE ALSO

vgcreate(1M), vgextend(1M), vgchange(1M), vgdisplay(1M).

NAME

vgremove - remove definition of one or more volume groups from the system

SYNOPSIS

/etc/vgremove *volume_group_name ...*

DESCRIPTION

vgremove removes from the system the last physical volume of the volume group and the definition of the volume group or groups specified by *volume_group_name*. Since all system knowledge of the volume group and its contents are removed, that volume group can no longer be accessed.

All logical volumes residing on the last physical volume must be removed by executing **lvremove** before executing **vgremove**. (see *lvremove*(1M)).

vgremove is equivalent to the inverse of executing **vgcreate** for one physical volume (see *vgcreate*(1M)).

Before removing a volume group, two steps are necessary:

1. Remove all but one of the logical volumes belonging to the group by using **lvremove** command (see *lvremove*(1M)).
2. Remove the physical volumes belonging to the volume group by using **vgreduce** (see *vgreduce*(1M)).

If there is any phyical volume group created under *volume_group_name*, the physical volume group information is also removed from file **/etc/lvmpvg**.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **vgremove** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Remove volume group **/dev/vg02** from the system:

```
vgremove /dev/vg02
```

SEE ALSO

lvremove(1M), vgreduce(1M), vgchange(1M).

NAME

vgscan - scan all Physical Volumes looking for Logical Volume Groups

SYNOPSIS

`/etc/vgscan [-p][-v]`

DESCRIPTION

`vgscan` allows recreation of the `/etc/lvmtab` and possibly the associated Logical Volume Group device files. This command should be run only in the event of a catastrophic error such as the deletion of file `/etc/lvmtab` or the mismatch of names of the Physical Volumes in file `/etc/lvmtab` to the actual Physical Volume path configuration. If file `/etc/lvmtab` exists, the information contained in the file is used to assist in rebuilding the file, but the existing file is updated with the new corrected configuration.

`vgscan` searches each Physical Volume connected to the system, looking for Logical Volumes. It then groups these Physical Volumes into Volume Groups by matching the Volume Group information as found on the Physical Volumes. `vgscan` then searches the `/dev` directory for all group device files with the LVM Major number. It then tries to match device files with Logical Volume information as found on the Physical Volumes. If matches occur, the Volume Group name is determined from the device file path, and file `/etc/lvmtab` is updated with the Volume Group name and the list of Physical Volume Paths containing that Volume Group. For each Volume Group where the device files cannot be matched, the list of Physical Volumes for each Volume Group is printed. The `vgimport` command should then be run on each set of Physical Volumes (see *vgimport*(1M)).

Options

`vgscan` recognizes the following options:

`-p` Preview the actions taken but do not update file `/etc/lvmtab`. This option is best used in conjunction with the `-v` option.

`-v` Prints verbose messages.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, `vgscan` behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Scan all the Physical Volumes on the system:

```
vgscan -p -v
```

SEE ALSO

vgexport(1M), vgimport(1M).

NAME

vgsync - synchronize stale logical volume mirrors in one or more volume groups

SYNOPSIS

/etc/vgsync *volume_group_name ...*

REMARKS

This software requires installation of optional LVM MIRRORING software (not included in the standard HP-UX operating system) before it can be used.

DESCRIPTION

The **vgsync** command synchronizes the physical extents in each mirrored logical volume in the volume group specified by *volume_group_name*. Synchronization occurs only on the physical extents that are **stale** mirrors of the original logical extent.

The synchronization process can be time consuming, depending on the hardware characteristics and the amount of data. Unless disabled, the mirrors within a volume group are synchronized automatically when the volume group is activated by the **vgchange -a y** command.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **vgsync** behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

Synchronize the mirrors on volume group **/dev/vg04**:

```
vgsync /dev/vg04
```

SEE ALSO

lvsync(1M), vgchange(1M), vgdisplay(1M).

NAME

vhe_altlog - login when Virtual Home Environment (VHE) home machine is not available

SYNOPSIS

`/usr/etc/vhe/vhe_altlog`

DESCRIPTION

`vhe_altlog` is a shell script that permits a user to log in when the home machine is not accessible through Virtual Home Environment (VHE). This script is executed when a login using the user name of *altlogin* is completed. After the user logs in using the user login name `altlogin`, `vhe_altlog` asks for a user name and password. If these are valid, the user is logged in on the machine and the home directory is a temporary directory such as `/tmp`. This provides user access to other machines in the group of VHE nodes even though a home machine is not available.

A user entry for `altlogin` must be present in `/etc/passwd` for it to be a valid login name. A typical entry resembles the following:

```
altlogin::6:1::/tmp:/usr/etc/vhe/vhe_altlog
```

DIAGNOSTICS

If an invalid user name or password is supplied, the attempted login is rejected.

AUTHOR

`vhe_altlog` was developed by HP.

FILES

`/etc/passwd`

SEE ALSO

vhe_mounter(1M), vhe_u_mnt(1M), vhe_list(4).

NAME

vhe_mounter - start the Virtual Home Environment (VHE)

SYNOPSIS

`/usr/etc/vhe/vhe_mounter`

DESCRIPTION

vhe_mounter is a shell script that configures a machine to operate with the Virtual Home Environment (VHE). VHE enables users to have the same view of their execution environments when logging in on machines interconnected with VHE. Machines connected with VHE must also be running the Network File System (NFS).

Information needed by **vhe_mounter** is provided by file `/etc/vhe_list` which contains a list of host names included in the group of VHE machines.

DIAGNOSTICS

vhe_mounter always returns exit code 0.

AUTHOR

vhe_mounter was developed by HP.

FILES

`/etc/vhe_list`

SEE ALSO

vhe_altlog(1M), vhe_u_mnt(1M), vhe_list(4).

NAME

vhe_u_mnt - perform Network File System (NFS) mount to remote file system

SYNOPSIS

`/usr/etc/vhe/vhe_u_mnt`

DESCRIPTION

`vhe_u_mnt` enables a user to perform a Network File System (NFS) mount to a remote file system. `vhe_u_mnt` is executed upon completion of a login using user name `mounter`. After logging in as user `mounter`, `vhe_u_mnt` asks for the name of the machine to which an NFS mount is to be done. If that machine name is listed in file `/etc/vhe_list`, mounts that are valid for that machine are made. This prevents the command from giving a user the ability to do NFS mounts to arbitrary machines. File `/etc/vhe_list` contains a list of hostnames that are part of the VHE group.

User name `mounter` must be present in file `/etc/passwd` file for it to be a valid login name. A typical entry resembles:

```
mounter::6:1::/:/usr/etc/vhe/vhe_u_mnt
```

DIAGNOSTICS

If a machine name is supplied that is not contained in `/etc/vhe_list`, an error message is produced indicating that the machine is not on the list of machines available for mounting.

AUTHOR

`vhe_u_mnt` was developed by HP.

FILES

`/etc/passwd`
`/etc/vhe_list`

SEE ALSO

vhe_altlog(1M), vhe_mounter(1M), vhe_list(4).

NAME

vipw - edit the password file

SYNOPSIS

vipw

DESCRIPTION

vipw edits the password file while setting the appropriate locks, and does any necessary processing after the password file is unlocked. If the password file is already being edited, you will be told to try again later. The **vi** editor is used unless the environment variable **EDITOR** indicates an alternate editor. **vipw** performs a number of consistency checks on the password entry for **root**, and does not allow a password file with an incorrectly formatted root entry to be installed.

WARNINGS

An **/etc/ptmp** file not removed when a system crashes prevents further editing of the **/etc/passwd** file using **vipw** after the system is rebooted.

AUTHOR

vipw was developed by the University of California, Berkeley.

FILES

/etc/ptmp

SEE ALSO

passwd(1), passwd(4).

NAME

volcopy,labelit - copy file systems with label checking

SYNOPSIS

/etc/volcopy [*options*] *fsname special1 volname1 special2 volname2*

/etc/labelit *special* [fsname volume [**-n**]]

DESCRIPTION

volcopy makes a literal copy of the file system using a block size matched to the device.

Options

volcopy recognizes the following options and command-line arguments:

-a invoke a verification sequence requiring a positive operator response instead of the standard delay before the copy is made.

-s (default) invoke the DEL if wrong verification sequence.

Other options are used with 9-track magnetic tapes:

-bpi *density* bits per inch

-feet *size* size of reel in feet

-reel *num* beginning reel number for a restarted copy

-buf use double buffered I/O

volcopy requests length and density information if this is not given on the command line or if it is not recorded on an input tape label. If the file system is too large to fit on one reel, *volcopy* prompts for additional reels. Labels of all reels are checked. Tapes can be mounted alternately on two or more drives. If **volcopy** is interrupted, it asks if the user wants to quit or wants to escape to the command interpreter. In the later case, other operations (such as **labelit**) can be performed before returning to **volcopy** by exiting the command interpreter.

The *fsname* argument represents the file system name on the device (e.g., root) being copied.

special should be the physical disk section or tape (e.g., **/dev/rdsk/1s3** or **/dev/rmt/0m**).

volname is the physical volume name; it should match the external sticker. Such label names ere limited to six or fewer characters. The argument *volname* can be - to use the existing volume name.

The arguements *special1* and *volname1* are the device and volume, respectively, from which the copy of the filesystem is being extracted. The arguments *special2* and *volname2* are the target device and volume, respectively.

The command **labelit** can be used to provide initial labels for unmounted disk or tape file systems. With the optional arguements omitted, **labelit** prints current label values. The **-n** option provides for initial labeling of new tapes only (this destroys previous contents).

NAME

vtdaemon - respond to vt requests

SYNOPSIS

vtdaemon [**-g**[*ngateway*]] [**-n**] *lan_device lan_device* ...

DESCRIPTION

vtdaemon responds to requests from other systems (via local area network) made by *vt(1). vtdaemon* spawns a server to respond to each request that it receives.

The **-g** option causes *vtdaemon* to rebroadcast all requests received on one lan device to all other lan devices specified on the command line. The optional parameter *ngateway* specifies the maximum number of *vtgateway* servers that can be in operation concurrently. If *ngateway* is not specified, there will be no limit on the number of vtgateway servers that can be in operation concurrently.

The **-n** option causes *vtdaemon* to ignore all requests that have come through a gateway.

The remaining arguments are the full path names of lan devices that *vtdaemon* looks for requests on. If no lan devices are specified, the default lan device is used. The major number for this device must correspond to a IEEE802.3 local area network device.

Another function of *vtdaemon* is to create *portals* and service portal requests. A *portal* is a callout device that can be used by *uucico(1M)* to communicate to another machine via local area network. Portals are created by *vtdaemon* according to the configuration information found in the file **/usr/lib/uucp/L-vtdevices**. Each line in **L-vtdevices** has the format:

<calldev>[,<lan device>] <nodename>

For each line, *vtdaemon* creates a portal named *calldev* in **/dev**. Whenever this device is opened, *vtdaemon* spawns a server that creates a connection to the system specified by *nodename* via the lan device specified. If no lan device is specified, the first one specified on the command line when *vtdaemon* was started is used (or the default lan device is used if no lan devices were specified on the command line).

vtdaemon should be terminated by sending signal **SIGTERM** to it. When *vtdaemon* receives this signal it removes all of the portals it created in **/dev** before exiting.

FILES

/etc/vtdaemonlog	logfile used by *vtdaemon*.
/dev/ieee	default lan device name.

SEE ALSO

uucico(1M), vt(1).

The *vt* tutorial in *Remote Access Users Guide*.

DIAGNOSTICS

Diagnostics messages produced by *vtdaemon* are written to **/usr/contrib/lib/vtdaemonlog**.

WARNINGS

vtdaemon uses the Hewlett-Packard **LLA** (Link Level Access) direct interface to the HP network drivers. *vtdaemon* uses the multicast address **0x01AABBCCBBAA**. It should not be used or deleted by other applications accessing the network. *vtdaemon* uses the following **IEEE 802.3** *sap* (service access point) values: **0x90**, **0x94**, **0x98**, **0x9C**, **0xA0**, **0xA4**, **0xA8**, **0xAC**, **0xB0**, **0xB4**, **0xB8**, **0xBC**, **0xC0**, **0xC4**, **0xC8**, **0xCC**, **0xD0**, and **0xD4**. They should not be used by other applications accessing the network.

NAME

wall, cwall - write to all users

SYNOPSIS

`/etc/wall` [`-g`*groupname*] [*file*]

`/etc/cwall` [`-g`*groupname*] [*file*]

DESCRIPTION

`wall`, when invoked without arguments, reads the standard input until an end-of-file. It then sends this message to all currently logged-in users preceded by:

```
Broadcast Message from ...
```

If the `-g`*groupname* option is specified, `wall` sends the message to all currently logged-in *groupname* members (as specified in `/etc/group`) preceded by:

```
Broadcast Message from  ...  to group groupname
```

If *file* is specified, `wall` uses *file* as its standard input.

`wall` is used to warn all users, typically prior to shutting down the system.

In the HP Clustered environment, `cwall` can be used to write to all users in the cluster.

The sender must have appropriate privileges to override any protections the users may have invoked (see *mesg*(1)).

`wall` has timing delays, and takes at least 30 seconds to complete.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

`Cannot send to ...`

The open on a user's tty file failed.

AUTHOR

`wall` was developed by AT&T and, HP.

FILES

`/dev/tty*`

SEE ALSO

mesg(1), write(1).

STANDARDS CONFORMANCE

`wall`: SVID2, XPG2, XPG3

NAME

whodo - which users are doing what

SYNOPSIS

`/etc/whodo`

DESCRIPTION

whodo produces merged, reformatted, and dated output from the **who** and **ps** commands (see *who*(1) and *ps*(1)).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

If **LC_COLLATE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **whodo** behaves as if all internationalization variables are set to "C" (see *environ*(5)).

FILES

`/etc/passwd`

SEE ALSO

ps(1), who(1).

STANDARDS CONFORMANCE

whodo: SVID2

(Requires Optional X.25 Software)

NAME

x25check, x25server - test connectivity between local and remote X.25 nodes

SYNOPSIS

/etc/x25check

/etc/x25check *x121_addr* [**-i** *prog_access_name*] [**-n** *num_packets*] [**-s** *size_of_packets*]

/etc/x25server

DESCRIPTION

x25check tests the connectivity of an X.25 interface up to the programmatic level. **x25check**, runs interactively when executed without any parameters and options. If parameters and options are specified, **x25check** runs without user interaction. **x25check** operates in conjunction with the **x25server** daemon to perform the test.

The test can be performed as a self-test, or in conjunction with a remote node. When performing a selftest, the designated X.25 interface must be connected to a X.25 packet switched data network, and the X.121 address specified is the address of the local X.25 interface card. If the test is to be performed in conjunction with a remote node, the X.25 interface card can be in any valid configuration (connected to a PSN or in a back-to-back configuration, but an **x25server** daemon must be running on the remote node. Note, that the remote node can in fact be another X.25 interface card on the local host.

During the test, **x25check** establishes a VC connection with the **x25server** daemon on the remote host and transmits the specified number of data packets of the specified size and closes the VC. In self-test mode the local **x25server** daemon communicates with **x25check**.

The **x25server** daemon is typically started in the **/etc/netlinkrc** file.

Parameters

x25check recognizes the following options and command-line arguments:

x121_addr specifies the X.121 address of the node to which *x25check* establishes a virtual circuit connection. It can be the address of either a remote node or another interface card on the local host, or if self-test is being performed, the address of the interface card on your local node. This parameter is required if an option is specified. *x121_addr* can be specified in the command line or interactively.

The following *x25check* options can be specified on the command line, but only the *prog_access_name* can be specified interactively.

-i *prog_access_name*
The X.25 programmatic access name as defined by the **x25init** command (see *x25init*(1M)). **Default:** The interface card referenced by **/dev/x25_0**.

-n *num_packets* Specifies the number of packets of data that you choose to send. **Default:** If **-s** is specified 1, otherwise 0.

-s *size_of_packets* Specifies the number of octets contained in each packet. The value must be less than the maximum packet size of 65535. If this parameter is specified, the default for **-n** becomes 1. **Default:** 128 octets.

DIAGNOSTICS

If the node to which a message is sent is unable to receive or respond to the message, the network provider rejects the call packet and sends a clear packet to the originator. The cause code and diagnostic code from the clear packet is displayed along with a diagnostic message.

AUTHOR

x25check was developed by HP.

SEE ALSO

x25stat(1), x25init(4).

Installing and Administering X.25/9000.

(Requires Optional LAN/X.25 Software)

NAME

x25init - configures and initializes an X.25 interface card

SYNOPSIS

x25init [**-c** *config_file*] [*X.121_addr*] [**-d** *device_file*]
[**-ip** *IP_addr* [*-subnet subnet_mask*]] [**-n** *prog_access_name*]
[**-a** *ip_map_file*] [**-v**]

DESCRIPTION

x25init initializes an X.25/9000 interface card based on the contents of a configuration file. *sam*(1M) can optionally be used to create a configuration file. *x25init_smpl*(4) contains an example configuration file. If *x25init* detects that the interface card is in an error state (the software has detected a serious error), it executes *x25upload*(1M) prior to initializing the interface card.

A separate *x25init* command can be used to initialize the IP to X.121 address map table. The address map table is built by *sam*(1M).

Options

Options to the *x25init* command can be specified in any order. Although none of the options is specifically required, using *x25init* with no options is semantically invalid.

-c *config_file*
Specifies a configuration file to be used for initialization. *x25init* reads the file and initializes the X.25 interface card's data structures based on the file contents.

If the **-c** is not specified, *x25init* uses default values for all configuration parameters except those specified in the runstring. The default values for the X.25 configuration parameters are described in the *Installing and Administering X.25/9000* manual.

X.121_addr
Specifies an X.121 address for the X.25 interface card. The X.121 address is a string of up to 15 decimal digits. The X.121 address for your node can be obtained from your X.25 network provider.

A valid X.121 address must be given in the *x25init* command or in the specified configuration file.

-ip *IP_addr*
Specifies an IP address for your X.25 interface card if IP is to be used over X.25. The IP address must be specified in standard dot notation: *n.n.n.n*. *n* is a number from 0 to 255.

If no IP address is set, the X.25 interface being configured cannot be used to support ARPA/Berkeley Services or other Internet services. Refer to the *Installing and Administering X.25/9000* manual for more information about IP addresses.

-subnet *subnet_mask*
Specifies the optional subnet mask for your system if IP is to be used over X.25. The subnet mask must be specified in standard dot notation: *n.n.n.n*. *n* is a number from 0 to 255.

A valid subnet mask is anything except 0.0.0.0 or 255.255.255.255. The subnet mask can be specified only when an IP address is used. Refer to the *Installing and Administering X.25/9000* manual for more information about subnetting.

-d *device_file*
Specifies the X.25/9000 subsystem management device file. The device file is typically placed in the **/dev** directory. The device file must be in the form **x25_***n* where *n* is a decimal number in the range of 0 to 15. The default device file name is **/dev/x25_0**.

-n *prog_access_name*
Specifies the programmatic access name which is used as a unique identifier for a X.25 interface card. The programmatic access name is a string of up to twelve characters. Any character can be used in the name. The default programmatic access name is **x25pgmaccess**.

-a *ip_map_file*
Specifies the file containing the IP to X.121 address map table. The IP map table is not associated with any one interface. The IP map table can be initialized

separately from the X.25 interface card at any time after the X.25 interface cards are initialized without shutting down X.25.

x25init initializes the kernel address map table based on the contents of the specified file. For the exact syntax of the *ip_map_file* refer to the *Installing and Administering* X.25/9000 manual.

-v Provides additional Cause and Action Messages in addition to Error Message for Problem Resolution.

DIAGNOSTICS

Refer to the *Troubleshooting X.25/9000* manual for a description of the diagnostics generated by this command.

AUTHOR

x25init was developed by HP.

SEE ALSO

sam(1M), x25stat(1M), x25stop(1M), x25upload(1M), x25init_smpl(4).

Installing and Administering X.25/9000,
Troubleshooting X.25/9000.

NAME

x25lbtest - Series 300/400 PDI interface card loopback self-test

SYNOPSIS

/etc/x25lbtest -d *dev_file* **-v**

DESCRIPTION

x25lbtest checks the PDI frontplane (HP36941A) and modem lines. In order to check the interface card and the data line receivers/drivers, the production loop-back connector must be plugged into the PDI frontplane connector. When the loopback test is activited, the PDI interface card transmits data that is echoed back to itself by way of the loop-back connector.

Options

x25lbtest recognizes the following options:

-d *dev_file* Specify the device file for the X.25/9000 Series 300/400 interface card that is to perform the loopback self-test.

-v (verbose) Display more information in error messages, including a CAUSE and corresponding ACTION for the user.

DEPENDENCIES

x25lbtest is supported on Series 300/400 only.

DIAGNOSTICS

Refer to the *Troubleshooting X.25/9000* manual for a description of the diagnostics generated by this command.

AUTHOR

x25lbtest was developed by HP.

SEE ALSO

x25init(1M), x25upload(1M),

Troubleshooting X.25/9000.

(Requires Optional LAN/X.25 Software)

NAME

x25stop - shut down an X.25/9000 interface card gracefully.

SYNOPSIS

`/etc/25stop -d` *dev_file*

DESCRIPTION

`x25stop` shuts down an X.25/9000 interface card gracefully. Following successful shutdown, the X.25/9000 interface card enters an inactive state. The data structures remain intact and `x25stat` can be used to print out information about the card.

All X.25 connections using this interface are broken. All X.25 programmatic access socket users receive an error if they have not closed the socket prior to shutdown. An ENETDOWN error is issued on the next `bind()` or `connect()` call when `x25stop` is executed. All TCP connections running across the network are aborted.

Options

`x25stop` recognizes the following options and command-line arguments:

`-d` *dev_file* Specifies the device file name of the interface card to be shut down. The `-d` parameter is required.

DIAGNOSTICS

Refer to the *Troubleshooting X.25/9000* manual for a description of the diagnostics generated by this command.

AUTHOR

`x25stop` was developed by HP.

SEE ALSO

x25stat(1M), x25init(1M).

Installing and Administering X.25/9000,
Troubleshooting X.25/9000.

(Requires Optional X.25 Software)

NAME

x25upload - dump X.25/9000 interface card memory into a file.

SYNOPSIS

/etc/x25upload -d *dev_file* **-o** *ofile* **-v**

DESCRIPTION

x25upload dumps an interface card's memory into a file. This memory dump is in a special format that is not readable by the user but is useful to HP service representatives in diagnosing problems with the X.25/9000 interface card on Series 300/400 and Series 700/800 machines.

Options

/etc/x25upload recognizes the following options:

-d *dev_file*	Specify the device file for the X.25/9000 interface card to be dumped.
-o *ofile*	Specify the name of file to which the card memory dump will be written.
-v	Verbose option produces more information in error messages including **CAUSE** and a corresponding user **ACTION**.

NOTES

x25upload may be executed by **x25init** if it detects a problem in the interface card. When **x25init** executes **x25upload**, the *ofile* file is located in directory **/etc/x25/dumps**. The name of the file is in the form of a date/timestamp (*MMMdd.hhmmss*) representing month day, hour, minute, and second when the interface card memory dump occurred (for example, **/etc/x25/dumps/dec31.115959**). Directory **/etc/x25/dumps** must exist when **x25init** is executed.

DIAGNOSTICS

Refer to the *Troubleshooting X.25/9000* manual for a description of the diagnostics generated by this command.

DEPENDENCIES

The **/etc/x25/dumps** directory is distributed as part of the X.25/9000 link products.

AUTHOR

x25upload was developed by HP.

NOTES

If the **x25init** could not or did not schedule **x25upload**, initialization continues and the previous state of the interface card is lost when the code is downloaded to the card.

The *ofile* produced by this command is useful to HP field service personnel. You may be asked to supply this file when troubleshooting the X.25/9000 interface card and its software.

FILES

/etc/x25/dumps/

directory to which the interface memory card dump is written when **x25init** calls **x25upload** if the interface card is not functioning properly.

SEE ALSO

x25init(1M).

Troubleshooting X.25/9000.

NAME

x29printd - remote PAD printer server for LP requests over X.25 network

SYNOPSIS

/etc/x29printd [**-l** *loglevel*] [**-w** *maxwrap*] [**-v**]

DESCRIPTION

x29printd provides host access to remote PAD printers on X.25 networks. **x29printd** and Packet Assembler/Disassembler (PAD) devices use the CCITT X.3 and X.29 standards for communication protocol. With **x29printd**, users on HP 9000 host systems with X.25 access can send print requests to be printed on selected printers connected via PAD on X.25 networks. **x29printd** interfaces with the underlying X.25 software via the BSD sockets. It initiates X.25 calls and sends print data to the remote PAD. **x29printd** uses pty device pairs (see *pty*(7)) to communicate with the HP-UX line-printer spooling system. The HP-UX line-printer spooler writes data to a slave device and **x29printd** reads data from the corresponding master device. **x29printd** is thus basically a pipe through which data is passed from the pty to X.25.

x29printd can be invoked at system initialization time from the **/etc/netlinkrc** file. To ensure proper functionality, the X.25 subsystem should be initialized before invoking **x29printd**. **x29printd** must work with the HP-UX line printer spooler to prevent intermixed listings, to provide control of printout routing, and to allow users to cancel, restart, and adjust the priority of print requests. The remote PAD printers and the device interfaces must be configured in the HP-UX line printer spooler system.

Options

x29printd recognizes the following options:

-l *loglevel* Set the logging level for error messages from the **x29printd** daemon to logfile **/usr/adm/x29/x29printd/x29printd.log**. *loglevel* can be one of the following values:

- **0** No logging.
- **1** Minimum session logging and error logging.
- **2** PAD and X.25 network error logging plus **1**.
- **3** Information and status logging plus **2**.

Default is **1**.

-w *maxwrap*
Specify maximum size for server daemon logfile. Allowable range for *maxwrap* is 1-5000. Default is no maxwrap (if the **-w** option is missing and *loglevel* is 1, 2, or 3, the daemon logfile grows indefinitely).

-v
The **-v** option turns verbose on. Verbose is off when this option is not present. When verbose is on, explicit CAUSE of the error condition and the ACTION required for the error condition is displayed in the logfile.

Configuration

Configuration of remote PAD printers for use with **x29printd** is provided by two files, **/etc/x25/x29hosts**, and **/etc/x25/x3config**.

/etc/x25/x29hosts defines the configuration for **x29printd** for each remote PAD printer. A remote PAD printer configuration entry is identified by the keyword **printer** followed by information about the printer beginning with **{** and ending with **}**. The information required between the opening and closing braces specifies the device file to be used by the line printer spooler system, the local X.25 programmatic access name for call set-up, the X.121 address of the remote PAD printer, the reverse charge option for X.25 calls, the logging level on a per-call basis, and the X.3 configuration set name. A typical remote PAD printer entry in **/etc/x25/x29hosts** resembles:

```
printer{
    device          printer1
    name            hptndxk0
    x3              hp_printer
    remote_x121     408555111201
    reverse_charge  enable
    logging             1
}
```

`/etc/x25/x3config` provides the initial X.3 parameters for the remote PAD printers defined in `/etc/x25/x29hosts`. The example above uses the `hp_printer` set name to specify the X.3 parameter set in the `/etc/x25/x3config` file. An example of the `hp_printer` set is:

```
hp_printer {
        1     0
        2     0
        3     0
        4    10
        5     1
        6     0
        7     0
        8     0
        9     0
       10     0
       11    14
       12     1
       13     0
       14     0
       15     0
       16     8
       17    24
       18     0
       19     1
       20     0
       21     0
       22     0
}
```

The `x29printd` daemon should be invoked after the configuration for all remote PAD printers is set-up in `/etc/x25/x29hosts` and `/etc/x25/x3config`.

After invoking `x29printd`, the remote PAD printer must also be configured in the spooler system for remote printing to work with the HP-UX line printer spooler commands. The command to configure a remote PAD printer in the spooler system is `lpadmin -p` *printer_name* `-v` *device_file_name* `-m` *printer_model* where

printer_name is the name given to this printer at the user level. It is the value that the user would give for the `-d` option in the `lp` command when a file is to be printed (see *lp*(1)).

device_file_name is `/dev/x29/`*device*`name`. The value for device_name is the same as that entered in the `device` entry in `/etc/x25/x29hosts`.

printer_model is the HP printer model name. Examples are `thinkjet`, `laserjet`, `hp2563a`, and so forth.

The `lpadmin` command associates a printer name with a device file name and `/etc/x25/x29hosts` associates the device file name with the remote printer's X.121 address (see `lpadmin`(1)). `lp` can then be used for printing to remote PAD printers.

The `x29printd` daemon forks a child for each one or more consecutive `lp` requests received from the line printer spooler system. The daemon continues waiting for requests from the line printer spooler system while the child transmits data to the remote PAD printer. After all print requests are sent to the remote PAD printer, the child clears the virtual circuit and terminates.

EXTERNAL INFLUENCES

`x29printd` is implemented with native language support. The logging can be in a foreign language by setting the `LANG` environment variable to correct NLS values and putting corresponding message catalog files in `/usr/lib/nls/`. See *nlsinfo*(1).

DIAGNOSTICS

`x29printd` exits if error conditions exist and cannot be corrected at start-up.

`x29printd: Must have root capability to start server.`
`x29printd` can be invoked only by the super-user.

`Error accessing the configuration file.`
Unrecoverable errors exist either in `/etc/x25/x29hosts` or in `/etc/x25/x3config`. Correct the errors in the files and start `x29printd` again.

`Open error on all master devices`
`Open error on all slave devices`
`x29printd` could not allocate pty master slave device pairs. Examine the state of the pty devices, correct any discrepancies between the master and slave pty devices and try again.

`x29printd` does not exit if errors occur on specific print requests or X.25 connections to remote PAD printers. Errors of this nature are logged to the logfiles in the directory `/usr/adm/x29/x29printd`. The daemon logfile is `x29printd.log`. There are also logfiles for `x29printd children forked by the daemon.` The logfiles for the children are identified by the device name in `/etc/x25/x29hosts` followed by the process ID of the child. The daemon forks a child for each X.25 connection established.

Error messages in either the daemon or the child logfiles are timestamped and identified by a system error code, a `x29printd` error code, and a brief error message. The format of the error message is:

<date> <time> <system error code>.<x29printd error code> <error message>

A typical error message resembles:

```
03/10/92 12:30:37 239.2303 Unable to connect to remote node
```

which says error code 239 was returned when `x29printd` tried to establish a connection to the remote PAD printer. Error 239 in `/usr/include/sys/errno.h` is ECONNREFUSED which means the connection request is refused by the remote PAD.

WARNINGS

`x29printd` has no spooling capability. It does not enable, disable, accept, or reject printers. It does not schedule or cancel print requests.

If a new `x29printd` is invoked when a previous `x29printd` daemon already exists and is running on the host system, the existing `x29printd` is killed before the new `x29printd` becomes a daemon.

`x29printd` supports 250 remote PAD printers. There must be one pty master-slave device pair for each remote PAD printer configured. The X.25 subsystem must have enough virtual circuits available to allow one virtual circuit for each PAD printer being used.

`x29printd` does not support permanent virtual circuits. It can only be used on switched virtual circuits.

`x29printd` does not support the fast-select feature in X.25.

AUTHOR

`x29printd` was developed by HP.

FILES

`/etc/x25/x29hosts`
`/etc/x25/x3config`

SEE ALSO

enable(1), lp(1), lpstat(1), mail(1), slp(1), x25stat(1), accept(1M), lpadmin(1M), lpsched(1M), mklp(1M), rcancel(1M), rlp(1M), rlpdaemon(1M), rlpstat(1M), sam(1M), x25init(1M), x29hosts(4), x3config(4).

Installing and Administering X.25/9000.
Troubleshooting X.25/9000.
HP 9000 Series 800 System Administration
HP 9000 Series 300/400 System Administration
HP 9000 Series 700 System Administration

STANDARDS CONFORMANCE

The implementation of X.29 and X.3 protocols in `x29printd` conforms to the 1984 CCITT standards.

(Requires Optional LAN/X.25 Software)

NAME

x29server - X.29 PAD support server

SYNOPSIS

/etc/x29server [**-l** *loglevel*] [**-w** *maxwrap*]

DESCRIPTION

x29server provides PAD support for the X.25/9000 link. The server (a user level process) accesses the X.25 level 3 via BSD sockets (programmatic interface). The server listens for call requests on any X.25 interface. When it receives a request from a PAD, **x29server** checks the calling address for access security. If valid, the call is accepted.

After call setup, **x29server** can receive two types of data: Normal terminal traffic, and PAD control messages. Terminal traffic is passed to pseudo-terminal drivers (see *pty*(7)). PAD control messages are processed by the server.

In the other direction, **x29server** passes data from pseudo-terminal drivers to X.25 level 3. It also processes requests by applications to alter tty parameters. These requests are mapped to X.3 parameters and sent to the remote PAD terminal.

The server is basically a pipe through which data is passed from X.25 to the pty, and vice versa. Terminal data is not modified.

Options

x29server recognizes the following options:

-l *loglevel* Set the logging level for log messages from **x29server** daemon to the logfile **/usr/adm/x29/x29server/x29server.log**. *loglevel* can be one of the following values:

- **0** No logging.
- **1** Minimum session logging and error logging.
- **2** PAD and X.25 error and warning logging plus **1**.
- **3** Information and status logging plus **2**.

Default is **1**.

-w *maxwrap*
Specify maximum size for server daemon logfile. Allowable range for *maxwrap* is 1-5000. Default is no maxwrap.

Configuration

System access security with **x29server** is provided by the **/etc/x25/x29hosts** file. A PAD-support server entry in **/etc/x25/x29hosts** is identified by the keyword **pad_spt** followed by information on access security, X.3 configuration, logging level, and reverse charging beginning with **{** and ending with **}**.

Incoming calls have their calling X.121 addresses compared to the **remote_x121** address in all **pad_spt** entries in **/etc/x25/x29hosts**. If there is a match, the call is accepted. The legal character set for **remote_x121** includes digits, from 0 through 9, the character **F**, the question mark **?**, and the asterisk *****.

Exact-address matching is provided with digits 0 through 9. The special-address **FFFFFFFFFF** matches to the null calling addresses. Wildcard-address matching is provided with **?** (for any single digit) and ***** (for any address). If none of the **remote_x121** addresses match the calling address, the virtual circuit is cleared immediately, not granting system access to the remote PAD user.

A typical PAD support server entry in **/etc/x25/x29hosts** resembles:

```
pad_spt{
     remote_x121      408555120801
     x3               hp_padsrvr
     reverse_charge   disable
     logging          1
}
```

/etc/x25/x3config provides the initial X.3 parameters for the incoming connections from the PAD. The example above uses **hp_padsrvr** to specify the X.3 parameter set in the **/etc/x25/x3config**

file. An typical **hp_padsrvr** set resembles:

```
hp_padsrvr {
    1          1          1
    2          1          1
    3         94        127
    4          0          0
    5          1          1
    6          5          5
    7         21         21
    8          0          0
    9          0          0
    10         0          0
    11        14         14
    12         1          1
    13         0          0
    14         0          0
    15         1          0
    16         8          8
    17        24         24
    18         0          0
    19         1          1
    20         0          0
    21         0          0
    22         0          0
}
```

x29server should be invoked after system access security and initial X.3 configurations are set up in **/etc/x25/x29hosts** and **/etc/x25/x3config**.

x29server can be started at system initialization time from **/etc/netlinkrc**. To ensure proper functionality, the X.25 subsystem should be initialized before invoking **x29server**. To start **x29server**, use the following command:

/etc/x29server

This starts the server background daemon which listens for all inbound PAD connections. If a PAD call is accepted, the server forks a child to handle the new connection. The parent continues to listen for incoming calls.

To stop the parent server, use the following command:

kill -9 *process_id_of_server_daemon*

Incoming calls are no longer accepted. To stop a child **x29server**, replaces the process ID of the server with the process ID of the child.

Note: A child process normally terminates at the end of a login session as a result of the logout process. The call is cleared and the child process exits. A child can also be terminated by an Invitation-to-Clear command from the remote PAD.

EXTERNAL INFLUENCES

x29server is implemented with native language support. The logging can be in a foreign language by setting the environment variable **LANG** to correct **NLS** values and putting corresponding message catalog files in **/usr/lib/nls/** (see *nlsinfo*(1)).

DIAGNOSTICS

Diagnositic messages for **x29server** are written to logfiles in the directory **/usr/adm/x29/x29server/**. The **x29server** daemon writes messages to **x29server.log** to log incoming call events, as well as events concerning the daemon server process itself.

The child inherits its logging level from the daemon if no logging level is specified in **/etc/x25/x29hosts** for the incoming call address. Otherwise, the child's logging level is that specified in **/etc/x25/x29hosts**. The child log filename is **/usr/adm/x29/x29server/x29log***xxxxx* where *xxxxx* is the process ID of the child.

(Requires Optional LAN/X.25 Software)

WARNINGS

PAD support security is provided by **/etc/x25/x29hosts**. If **/etc/x25/x29hosts** does not exist or is not accessible, all inbound calls are rejected.

x29server does not support permenant virtual circuits. **x29server** can only be used on switched virtual circuits.

x29server binds to all X.25 interfaces on the host system. It does not support binding to only a particular interface.

x29server does not support binary data transfer or block-mode applications.

x29server does not support X.28 local parameters on PAD.

x29server does not support fast-select facility features in X.25.

There must be one available pty configured on the system per desired PAD connection. The number of remote PAD users that **x29server** supports is determined by the number of available virtual circuits or psuedo-terminals, whichever is lower.

AUTHOR

x29server was developed by HP.

FILES

/etc/x25/x29hosts
/etc/x25/x3config

SEE ALSO

sam(1M), x25init(1M), x25stat(1), x29hosts(4), x3config(4).

Installing and Administering X.25/9000.
Troubleshooting X.25/9000.

STANDARDS CONFORMANCE

The implementation of X.29 and X.3 protocols in **x29server** conforms to the 1984 CCITT standards.

NAME

x29uucpd - PAD UUCP server for UUCP requests to remote hosts on X.25 network

SYNOPSIS

/etc/x29uucpd [**-l** *loglevel*] [**-w** *maxwrap*] [**-v**]

DESCRIPTION

x29uucpd provides UUCP connectivity on X.25 networks using CCITT Recommendations X.3 and X.29. With **x29uucpd**, users on HP 9000 host systems with X.25 access and UUCP can execute UUCP subsystem commands to other systems running X.25, PAD support, and UUCP. **x29uucpd** interfaces with the X.25 subsystem via BSD sockets to initiate Call Request packets to remote systems on X.25. When a Call Request arrives at a destination system, it is received by the PAD support application running on the destination system. On HP 9000 hosts, the PAD support application is **x29server**. **x29uucpd** interfaces with UUCP via the pty driver (see also *pty*(7)). The UUCP subsystem writes data to a slave device and **x29uucpd** reads data from the corresponding master device. **x29uucpd** is thus basically a pipe through which data is passed from the pty to X.25.

x29uucpd can be invoked at system initialization time from the **/etc/netlinkrc** file. To ensure proper functionality, the X.25 subsystem should be initialized before invoking **x29uucpd**. **x29uucpd** creates device files in **/dev/x29**. The device files must be configured in the UUCP subsystem for UUCP to interface with **x29uucpd**.

Options

x29uucpd recognizes the following options and command-line arguments:

-l *loglevel* Set the logging level for error messages from the **x29uucpd** daemon to logfile **/usr/adm/x29/x29uucpd/x29uucpd.log**. *loglevel* can be one of the following values:

- **0** No logging.
- **1** Minimum session logging and error logging.
- **2** PAD and X.25 network error and warning logging plus **1**.
- **3** Information and status logging plus **2**.

Default is **1**.

-w *maxwrap*
Specify maximum size for server daemon logfile. Allowable range for *maxwrap* is 1 through 5000. Default is no maxwrap. (If the **-w** option is missing and *loglevel* is 1, 2, or 3, the daemon logfile grows indefinitely.)

-v
The **-v** option turns verbose on. Verbose is off when this option is not present. When verbose is on, explicit CAUSE of the error condition and the ACTION required for the error condition is displayed in the logfile.

Configuration

Configuration of the UUCP destination systems for **x29uucpd** is provided by two files, **/etc/x25/x29hosts**, and **/etc/x25/x3config**.

/etc/x25/x29hosts defines the configuration for **x29uucpd** for each UUCP destination system. A UUCP PAD support configuration entry is identified by the keyword **pad_uucp** followed by information about the UUCP destination system beginning with { and ending with }. The information required between the open and close braces specifies the device file to be used by the local UUCP subsystem, the local X.25 programmatic access name for call set-up, the X.121 address of the remote UUCP destination, the reverse charge option for X.25 calls, the logging level on a per-call basis, and the X.3 configuration set name. An typical UUCP PAD support entry in **/etc/x25/x29hosts** resembles:

```
pad_uucp{
    device              x25uucp
    name                hptndxk0
    remote_x121         4085551113
    reverse_charge      enable
    x3                  hp_uucp
    logging             3
```

```
}
```

`/etc/x25/x3config` provides the initial X.3 parameters for the remote UUCP destinations defined in `/etc/x25/x29hosts`. The example above uses `hp_uucp` to specify the X.3 parameter set in the `/etc/x25/x3config` file. An typical `hp_uucp` set resembles:

```
hp_uucp{
        1           0
        2           0
        3           0
        4          10
        5           1
        6           0
        7           0
        8           0
        9           0
       10           0
       11          14
       12           1
       13           0
       14           0
       15           0
       16           8
       17          24
       18           0
       19           1
       20           0
       21           0
       22           0
}
```

The **x29uucpd** daemon should be invoked after the configuration for all UUCP destination systems is set up in `/etc/x25/x29hosts` and `/etc/x25/x3config`.

Several UUCP files must be set up correctly for the interoperation of UUCP and **x29uucpd**. These files reside in the `/usr/lib/uucp` directory. `Systems` contains login information for the remote UUCP hosts. `Permissions` specifies the access, send, read, write, and execute permissions for the remote UUCP hosts. `Devices` contains entries for device files and other device-related information for the remote UUCP hosts. In addition, the `/usr/spool/uucp/LCK..x29` directory must exist on the local host system.

The **x29uucpd** daemon forks a child for each **uucp** request received from **UUCP** (see *uucp*(1)). The daemon continues with waiting for requests from UUCP while the child transmits data between the local and remote systems. When the **uucp** request completes, the virtual circuit is cleared, and the child process exits.

EXTERNAL INFLUENCES

x29uucpd is implemented with native language support. The logging can be in a foreign language by setting the environment variable **LANG** to correct NLS values and putting corresponding message catalog files in `/usr/lib/nls`. See *nlsinfo*(1).

DIAGNOSTICS

x29uucpd exits if error conditions exist that cannot be corrected at start-up.

`x29uucpd: Must have root capability to start server.`
x29uucpd can be invoked only by the super-user.

`Error accessing the configuration file.`
Unrecoverable errors exist either in `/etc/x25/x29hosts` or in `/etc/x25/x3config`. Correct the errors in the files and start **x29uucpd** again.

`Open error on all master devices , Open error on all slave devices`
x29uucpd could not allocate pty master-slave device pairs. Examine the state of the pty devices, correct any discrepancies between the master and slave pty devices and try again.

x29uucpd does not exit if errors occur on specific **uucp** requests or X.25 connections to remote UUCP destinations. Errors of this nature are logged to the logfiles in the directory **/usr/adm/x29/x29uucpd**. The daemon logfile is **x29uucpd.log**. There are also logfiles for x29uucpd children forked by the daemon. Logfiles for the children are identified by the device name in **/etc/x25/x29hosts** followed by the process ID of the child. The daemon forks a child for each X.25 connection established.

Error messages in either the daemon or the child logfiles are timestamped and identified by a system error code, a **x29uucpd** error code, and a brief error message. The format of the error message is

<date> <time> <system error code>.<x29uucpd error code> <error message>

An example of an error message is:

```
03/10/92 12:30:37 239.2303 Unable to connect to remote node
```

which says error code 239 was returned when **x29uucpd** tried to establish a connection to the remote UUCP system. Error 239 in **/usr/include/sys/errno.h** is ECONNREFUSED which means the connection request is refused by the remote PAD support server.

WARNINGS

If a new **x29uucpd** is invoked when a previous **x29uucpd** daemon already existed and was running on the host system, the previous **x29uucpd** is killed before the new **x29uucpd** becomes a daemon.

x29uucpd supports 250 remote UUCP systems. There must be one pty master-slave device pair for each remote UUCP destination configured. The X.25 subsystem must have enough virtual circuits available to allow one virtual circuit for each remote UUCP transfer in progress.

x29uucpd does not support permanent virtual circuits. It can only be used on switched virtual circuits.

x29uucpd does not support the fast-select feature in X.25.

AUTHOR

x29uucpd was developed by HP.

FILES

/etc/x25/x29hosts
/etc/x25/x3config

SEE ALSO

uucp(1), mail(1), uux(1), sam(1M), x25init(1M), x25stat(1), x29server(1M), x29hosts(4), x3config(4).

UUCP HP-UX Concepts and Tutorials.
Installing and Administering X.25/9000 ,
Troubleshooting X.25/9000 .

STANDARDS CONFORMANCE

The implementation of X.29 and X.3 protocols in **x29uucpd** conforms to the 1984 CCITT standards.

NAME

xstm - X11-based support tool manager

SYNOPSIS

/usr/diag/bin/xstm [**-m**] [**-l** *log_file*]

DESCRIPTION

xstm provides access to a variety of system hardware support tools through an X Windows graphical user interface. Note that when you run this program for the first time, start-up initialization takes longer than when it is run again. In this sense, the initial execution of **xstm** is the equivalent of using the **-m** option (discussed below).

xstm presents the machine configuration with a system device map framed in a main window with a pull-down menu bar along the top. A descriptive icon is presented in the device map for each interface card and device detected on the system. Devices are depicted in a hierarchical, inverted "tree" format; the "root" is the host system ID itself. Beneath the host icon the device map tree breaks out to the system's I/O and system interface devices, then to individual system devices. Subcomponents of the System Processor Unit (SPU) that can be individually identified (such as memory or floating point processor) are presented to the sides of the "host" unit.

General application functions such as viewing the application log file, loop control, and getting help are available from the main window pulldown menu bar.

Device support functions can be executed through pop-up menus activated by pressing the left pointer button with the pointer positioned over the device's representative icon. Actions that can be performed on the specified device depend on what support functions are available on the system for that device type. Typically, these actions are **verify**, **exercise**, and **diagnose**. **information** about the device, obtained by the system mapping facility, is available for all devices. Results (success, failure, warning) are indicated by changes to the color (or grey shade) of the device icon.

When an action is selected the device icon's color changes to pale blue, indicating that an action is in progress. When the action has completed, the device icon's color signifies the success (green) or failure (red) of the action. Caution states (yellow) are indicated in some cases where indeterminate results are found or special precautions are required when dealing with the device. Detailed information on the results of the action can be found by invoking the **information** action, or by viewing the session log file that is maintained by **xstm**.

Some actions may require user intervention, such as mounting a tape and making sure that tape drive is on line. When such operator intervention is required, a dialog window appears on the display, with instructions on what action is necessary.

Options

-m At program start-up, force a search of physical devices on the system, and of diagnostic programs supported. When this option is used, **xstm** takes longer to reach the point where user interaction begins. This option is required when system configuration is changed or when a new diagnostic program is installed through **sysdiag**.

-l *log_file* Specifies the name of the file to which log events that occur during the time the application is active are written. Default is **./stm.log**.

MENUS

Main Window Menu Bar Functions

General application functions are available from pull-down menus accessed by the Main Window Menu Bar. To access a menu, position the pointer over the Menu Bar label to be activated, and press the left button. Drag the pointer down to the menu item to be selected, then release the button.

Actions

The Actions Menu provides access to general application functions that are device-independent. The following "action" functions are available:

Verify All Run the **verify** action on all system devices that can be verified (i.e., device types for which a verifier tool is available).

Cancel All cancels all device support functions that are currently active or pending.

Exit causes **xstm** to shut down and exit. If there are any active support processes, prompts to end these processes are allowing the user to let them complete before exiting, or cancel the exit request.

View

The View Menu can be used to display the contents of the main log file maintained by **xstm**.

Options

The Options Menu accesses optional user functions. Currently, the following option is available via this menu:

Looping provides loop control over device support tools being executed. This option is intended primarily for troubleshooting intermittent problems with system exercisers. When this pulldown menu is selected, you are prompted to select a looping style (i.e., loop by number of iterations, loop for a specified period of time in minutes, or loop infinitely), and to enter values appropriate for the style selected.

When a loop value is set, the loop value applies to all requested device support actions (verify, diagnose, and exercise) until the loop value is changed.

The following looping style options are available:

By Count	Loop a specific number of iterations.
By Time	Loop for a specific period of time, measured in minutes.
Forever	Loop on a test infinitely.

Looping styles are selected by positioning the pointer over one of the buttons and pressing the left button. The loop value box (below the buttons) changes to reflect the style selection.

After selecting the *count* or *time* looping style, enter the appropriate value in the loop value box at the bottom of the looping control panel. (*Loop Forever* does not require an entry in the loop value box). Initially, the default value of **1** is displayed in this window. In order to enter values for number of iterations or duration in minutes in the selection box, move the mouse pointer inside the selection box and press the left button to enter a value (this "selects" the value box). Enter a whole number greater than zero in the loop value box.

After entering the appropriate value in the selection box, set the chosen values by clicking on the **OK** widget, or reject them by clicking on the **CANCEL** widget.

The default loop value is "one iteration". If any loop value other than default is selected, the value is displayed on the device map to remind you what looping selection is in effect. To restore default looping, set looping back to looping style *count*, with a value of **1** iteration.

Note that if time expires in the middle of an operation's excution, the operation will complete. That is, *time* looping does not cause the operation to end prematurely or to abort.

HELP provides online information about **xstm**. A window with information about the selected topic is displayed. Press the window's **OK** button to close a help window. Topics available are:

On XSTM General information about the X11-based Support Tool Manager.

On Menu Bar Information about using the main window menu bar pulldown menus and the functions available from the pulldown menus.

On Device Map Help on the system device map, how to perform support functions, and how to get information about devices represented on the map.

On Miscellaneous Miscellaneous information about the product, including tips on using **xstm** and graphic user interface behaviour.

On Version Displays the version of **xstm** being used and the copyright declaration.

DEVICE ICON POP-UP MENU FUNCTIONS

Support functions for specific devices are available from pop-up menus associated with each device icon (graphic symbols representing devices, cards, and major system components) on the device map area.

To access a device action pop-up menu, position the pointer over the icon representing the device to be investigated and press the left button. Drag the pointer down to the menu item to be selected and release

the button.

Every device icon has at least one selection, **information**, that displays a window with device type and status information (see below). The other selections on the pop-up menu represent support functions that can be performed on the device. Selecting one of these menu items causes the respective function to be executed on the device or component.

Availability of other device action selections depends on whether or not support tools for the device type are installed on the system, and are detectable by the Support Tools Manager.

The device action selections are:

Information Pops up a Device Information window. This window displays the part number and part name information detected by the device mapping facility, and, where applicable, the status of device support functions that have been executed on the device.

The top of the window displays the hardware path, part number, and device description. If there are support functions available for the device, a radio button is displayed for each support function. Pressing one of these buttons causes the status of the respective support function for the device and, if the support tool provides one, the log file for the device to be displayed. Press the **OK** button to close the window.

The other selections which may be available are:

Diagnose Runs a diagnostic program on the device. Diagnostic programs are designed to detect and isolate faulty hardware on a device.

Exercise Exercisers continuously stress a device or subsystem. This function is useful in providing very high confidence verification and in detecting intermittent errors.

Verify Performs a simple test of component function, providing a "pass/fail" indication of device condition. "Verify" is typically the first-level test of a device's condition.

RESOURCES

xstm user-configurable application X11/Motif resources are in the file **Stm**. A base copy of this resource file is in directory **/usr/lib/X11/app-defaults/XStm**.

XStm contains the resources for device icon indication of device action results. These are set to colors by default. An example set of resources for grey scale is provided in a comment area of the file.

The bitmaps to use for representing various device types, as well as fonts and background colors, are also in the resource file.

EXTERNAL INFLUENCES

Environment Variables

DISPLAY must be set to the desired X Windows display ID (e.g., **local:0.0**).

WARNINGS

xstm is a tool that controls the execution of system- and device-level diagnostic and support tools. Executing this software places additional load on system throughput and sensitive system peripheral devices. **xstm** program uses considerable CPU time due to the nature of its internal processing. Unnecessary use of this product is strongly discouraged.

xstm (and **cstm/mstm**) are delivered configured so that any user can execute the "verify" tools. This can cause unnecessary system load should the program be executed indiscriminately. System administrators and managers may prefer to change the permission bits of the program so that only super-user can use the tool:

```
chmod 544 /usr/diag/bin/xstm
```

Most of the various dialog windows that appear in **xstm** are not delivered so that they "auto-raise" with the main window; this is to avoid having the window become unnecessarily obscured. If you are "missing" a window, try lowering the "top" window (or "shuffling" up or down). The missing window is probably underneath. In particular, if a Device Information window is already active, and you request the "information" action, the window doesn't auto-raise.

When you start **xstm** or **cstm** while someone else is running diagnostics on a device (e.g., SCSI disk), this device will not show up on the **xstm** or **cstm** device map. This is because the diagnostic has the device

for exclusive use; as a result the facility that maps system devices is not able to probe the device and treats it as non-existent.

Attempting to initiate a diagnose operation on **MEMORY** while someone else is already running the memory diagnostic program (whether through **xstm, cstm,** or **sysdiag**), the following error message results:

```
(MEMERR 10260) Memdiag unable to OPEN memory, BAD exit status = 501
```

This message does not imply a memory hardware problem, but is instead due to the limitation that only one diagnostic process is able to diagnose memory at any given time. This situation can also be encountered if the memory logging process (**MEMLOGP**) is scanning the memory controllers.

Occasionally, following a "Cancel Requests" request made from the "Actions" menu (on the main window pulldown menu bar), an error message stating **UNKNOWN CHILD SENT AM COMMAND** or **Reply is to non-existent request** may appear. This occurs when you cancel or abort a test, but the test result has already been issued. This condition is harmless and should be ignored in this circumstance.

When **xstm** has an execution error during startup, the error pop-up windows are not always displayed in their entirety before execution stops. In this condition, check the *standard error* port for your environment. Standard error can be directed to the X-terminal panel from which you entered the **xstm** command, or to an error log file such as **$(HOME)/.vue/Xerrors**.

If **xstm** fails during startup, the most common reason is that the program cannot open its log file **stm.log**. This can happen when the root user runs **xstm** from an NFS mounted file system, or when the log file permissions are set so that the **xstm** user has no write privilege.

Initiating exercise operations may impact system performance. In some cases, entering the diagnostic user interface (sysdiag) can result in the following error message:

```
*** ERROR - Could not establish communication with external parts of
    the diagnostic system (DUIINITERR 1)

*** Diagnostic system error 4435
```

In this situation, wait for the exercise operations to complete so that more resources can be shifted to the diagnostic system.

Do not run **xstm** or **cstm** from NFS-mounted current working directories. Running **xstm** or **cstm** from an NFS directory can lead to execution problems due to NFS security features and the execution mode of **xstm** and **cstm**.

Only one graphics verifier or exerciser can be run at a time. A program or system crash could prevent the graphics tool lock file from being removed when it should be. If a graphics verify or exercise cannot be started and no other test is being executed, removing the lock file **/tmp/.graphicsdaf.lock** should allow a single test to run.

When running the graphics exerciser for HP 98705A or HP 98705B, one of the two following entries should be in the **/usr/lib/X11/X0screens** file:

```
/dev/crt depth 16 doublebuffer      #for 16 planes (HP98705B)
/dev/crt depth 8                    #for 8 plane (HP98705A)
```

The graphics diagnostic causes an X11 (including HP-VUE) environment to reset when the diagnostic completes. This is because the diagnostic would otherwise leave the graphics device in an unknown state when the program finishes. When running from HP-VUE, the diagnostic in effect *logs you out* and returns to the *Vuelogin* screen. In other environments, when the diagnostic completes *you must manually* press [Ctrl-Shift-Reset] from the keyboard to return the graphics console to a normal operating state.

To run the graphics diagnostics for HP98705A, HP98705B (G98705DG) or HP98765A, HP98766A (G98735DG), make sure neither X11 windows nor HP-VUE are running on the ITE console. The graphics ITE should be exclusively accessed during the diagnostics test. After the tests are done, a [Ctrl-Shift-Reset] can restore the ITE console to its original state.

To test floppy disk drives, blank formatted media must be used. If the floppy is not formatted, the verification or exercise operation runs for a long period of time, then fails with the error message **Cannot read entire buffer. Read only 0 bytes instead of 4096 bytes.** If there is no media in the drive, the verification or exercise operation fails with the error message **Could not open temporary device file. Error: open (6; No such device or address) .**

Execution times for floppy disk device support operations take longer if the floppy disk device is unmounted. That is, verify/exercise/diagnose on a mounted floppy disk may execute in seconds, but these operations may take minutes if the device is unmounted.

When reporting problems on **xstm**, refer to part number B2478-10001.

AUTHOR

xstm was developed by HP.

FILES

XStm

Default X11 application resource definition file. Directories **/usr/lib/X11/app-defaults** and **$HOME** are searched to find **XStm**.

Session Log File

The session log (default name: **stm.log**) contains a detailed history of the actions performed by **xstm**. This log begins with a chart indicating the system and I/O configuration for the system and records the results for each action performed during the session. Each line of the chart specifies the location of the device, a description of the device, the current status of the diagnose action, and the current status of the verify action.

Note that the Diagnostic Status and Verifier Status columns can contain entries of **N/A** or **Not Checked**. **N/A** signifies that the Diagnose or Verify action is not available for the device. **Not Checked** signifies that the action has not been run, which is correct when **xstm** begins.

Other Files

/usr/diag/bin/am	support application manager
/usr/diag/bin/dtm	diagnostic tool manager
/usr/diag/bin/CXSTM000	default NLS message catalog

SEE ALSO

cstm(1M), mstm(1M).

NAME

ypinit - build and install Network Information Service databases

SYNOPSIS

/usr/etc/yp/ypinit -m [**DOM=***NIS_domain*]
/usr/etc/yp/ypinit -s *NIS_server_name* [**DOM=***NIS_domain*]

DESCRIPTION

ypinit is a shell script that creates Network Information Service (NIS) databases on either a master or slave NIS server. **ypinit** asks a few self-explanatory questions, and reports success or failure to the terminal. For an overview of Network Information Service, see *ypfiles*(4) and *ypserv*(1M).

Options

ypinit recognizes the following options and command-line arguments:

-m Create the local host as the master server to all maps (databases) provided in the NIS domain (see *domainname*(1)). All maps are built from scratch, either from information provided to **ypinit** at run-time, or from ASCII files in **/etc**. All such files should be complete and unabbreviated, unlike how they may exist on a NIS client machine (see *passwd*(4) for examples of abbreviated files).

See *ypmake*(1M) for more information on how NIS databases are built on the master server. Note that **ypinit** uses the **NOPUSH=1** option when invoking **ypmake**, so newly formed maps are not immediately copied to slave servers (see *ypmake*(1M)).

-s Create NIS databases on a slave server by copying the databases from an existing NIS server that serves the NIS domain.

The *NIS_server_name* argument should be the host name of either the master server for all the maps or a server on which the maps are current and stable.

DOM=*NIS_domain* Causes **ypinit** to construct maps for the specified *NISdomain*. **DOM** defaults to the NIS domain shown by the **domainname** command (see *domainname*(1).

DIAGNOSTICS

ypinit returns exit code 0 if no errors occur; otherwise, it returns exit code 1.

AUTHOR

ypinit was developed by Sun Microsystems, Inc.

FILES

/etc/group
/etc/hosts
/etc/netgroup
/etc/networks
/etc/passwd
/etc/protocols
/etc/rpc
/etc/services
/etc/vhe_list

SEE ALSO

domainname(1), makedbm(1M), vhe_altlog(1M), vhe_mounter(1M), vhe_u_mnt(1M), ypmake(1M), yppush(1M), ypserv(1M), ypxfr(1M), group(4), hosts(4), netgroup(4), networks(4), passwd(4), protocols(4), rpc(4), services(4), vhe_list(4), ypfiles(4).

NAME

ypmake - create or rebuild Network Information Service databases

SYNOPSIS

/usr/etc/yp/ypmake [DIR=*source_directory*] [DOM=*NIS_domain*] \
[NOPUSH=1] [PWFILE=*passwd_file*] [*map* ...]

cd /usr/etc/yp; make [DIR=*source_directory*] [DOM=*NIS_domain*] \
[NOPUSH=1] [PWFILE=*passwd_file*] [*map* ...]

DESCRIPTION

ypmake is a shell script in **/usr/etc/yp** that builds one or more Network Information Service (NIS) maps (databases) on a master NIS server. If no arguments are specified, **ypmake** either creates maps if they do not already exist or rebuilds maps that are not current. These maps are constructed from ASCII files. **yppush** is then executed to notify slave NIS servers of the change and make the slave servers copy the updated maps to their machines (see *yppush*(1M)).

If any *maps* are supplied on the command line, **ypmake** creates or updates those *maps* only. Permissible names for *maps* are the filenames in **/etc** listed under FILES below. In addition, specific *maps* can be named, such as **netgroup.byuser** or **rpc.bynumber**.

The **make** command can be used instead of *ypmake* (see *make*(1)). The **Makefile** in **/usr/etc/yp** calls the **ypmake** script to actually construct the maps. Better performance is achieved if **ypmake** is called directly, instead of via **make**.

Both the **Makefile** and **ypmake** script use four variables:

DIR=*source_directory*	The directory containing the ASCII source files from which maps are constructed. **DIR** defaults to **/etc**.
DOM=*NIS_domain*	Causes **ypmake** to construct maps for the specified *NIS_domain*. **DOM** defaults to the NIS domain shown by **domainname** (see *domainname*(1)).
NOPUSH=1	When non-null (null by default), **NOPUSH** inhibits copying the new or updated databases to the slave NIS servers. Only slave NIS servers in the specified *domain* receive **yppush** notification when **NOPUSH** is null.
PWFILE=*passwd_file*	Specifies the full pathname of the ASCII file that **ypmake** should use when building the NIS passwd maps. **PWFILE** defaults to **$DIR/passwd**.

The order of arguments passed to **ypmake** is unimportant, but the maps are built or updated in the left-to-right order provided.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of Network Information Service.

DIAGNOSTICS

ypmake returns one of the following exit codes upon completion:

0 Normal termination; no problems.
1 One or more unrecognized arguments were passed.
2 The NIS domain name is not set.
3 The subdirectory used to contain maps for a specific NIS domain, **/usr/etc/yp/domain_name**, does not exist or is not writable.
4 An error was encountered when building at least one of the maps.
5 One or more maps' ASCII files do not exist or are unreadable.

EXAMPLES

Create or rebuild the password databases (both the **passwd.byname** and **passwd.byuid** maps) from **/etc/passwd** and use **yppush** to copy the databases to any slave NIS servers in the default NIS *domain*:

```
ypmake passwd.byname
```

Create or rebuild the hosts databases from **/etc/hosts** but do not copy the databases to any slave NIS servers:

```
ypmake hosts NOPUSH=1
```

Create or rebuild the network maps from `/nis/sourcefiles/networks` and copy the maps to any slave NIS servers in NIS domain `DAE_NIS`:

```
ypmake DOM=DAE_NIS networks DIR=/nis/sourcefiles
```

AUTHOR

ypmake was developed by Sun Microsystems, Inc.

FILES

```
/etc/group
/etc/hosts
/etc/netgroup
/etc/networks
/etc/passwd
/etc/protocols
/etc/rpc
/etc/services
/etc/vhe_list
```

SEE ALSO

domainname(1), make(1), makedbm(1M), vhe_altlog(1M), vhe_mounter(1M), vhe_u_mnt(1M), ypinit(1M), yppush(1M), ypserv(1M), group(4), hosts(4), netgroup(4), networks(4), passwd(4), protocols(4), rpc(4), services(4), vhe_list(4), ypfiles(4).

NAME

yppasswdd - daemon for modifying Network Information Service passwd database

SYNOPSIS

/usr/etc/rpc.yppasswdd *passwd_file* [**-l** *log_file*] [**-m** [*arg1 arg2* ...]]

DESCRIPTION

The **yppasswdd** daemon handles password change requests from **yppasswd** (see *yppasswd*(1)). It changes a password entry in *passwd_file*, which must be in the format defined by *passwd*(4). The change is made only if the old password provided by **yppasswd** matches the encrypted password of that entry.

yppasswdd should be executed only on the master Network Information Service (NIS) server for the passwd database (map). The **yppasswdd** daemon is not executed by default, nor can it be started by **inetd** (see **inetd**(1M)). To enable automatic startup of **yppasswdd** at boot time, the **NIS_MASTER_SERVER** variable should be set to 1 in file **/etc/netnfsrc** on the master NIS server.

Options

yppasswdd recognizes the following options and command-line arguments:

-l *log_file* Log diagnostic and error messages to *log_file*. These messages are not available if **yppasswdd** is started without the **-l** option.

Information logged to the file includes date and time of the message, the host name, process ID and name of the function generating the message, and the message itself. Note that different services can share a single log file because enough information is included to uniquely identify each message.

-m [*arg1 arg2* ...] After *passwd_file* is modified, and if using the **-m** option, **yppasswdd** executes **ypmake** to update the NIS passwd database (see *ypmake*(1M) Any arguments following the **-m** flag are passed to **ypmake**.

To ensure that the passwd map is rebuilt to contain the new password and all slave NIS servers have their passwd maps properly updated to include the change, always use the **-m** option to **yppasswdd**, but do not use the **NOPUSH=1** argument to **ypmake**.

EXAMPLES

Assume the **yppasswdd** daemon is started on the master NIS server as follows:

```
/usr/etc/rpc.yppasswdd /etc/yp/src/passwd \
                       -l /usr/adm/yppasswdd.log \
                       -m passwd DIR=/etc/yp/src
```

This indicates that the ASCII file from which the NIS passwd database is built is **/etc/yp/src/passwd**. When this file is updated by a request from **yppasswd**, the NIS passwd database is rebuilt and copied to all slave NIS servers in the master's NIS domain (see *domainname*(1)).

Log messages are written to the file **/usr/adm/yppasswdd.log**.

WARNINGS

yppasswdd uses lock file **/etc/ptmp** to get exclusive access to *passwd_file* when updating it. The file **/etc/ptmp** may persist if *passwd_file* is being updated and

- The system crashes or
- **yppasswdd** is killed using **SIGKILL** (see *kill*(1) and *signal*(2)).

File **/etc/ptmp** must be removed before **yppasswdd** can function properly again.

vipw also uses **/etc/ptmp** when updating **/etc/passwd** (see *vipw*(1M)). As a result, **yppasswdd** competes with **vipw** when it updates *passwd_file* if *passwd_file* is **/etc/passwd**.

AUTHOR

yppasswdd was developed by Sun Microsystems, Inc.

FILES

/etc/ptmp lock file used when updating *passwd_file*

SEE ALSO
domainname(1), yppasswd(1), vipw(1M), ypmake(1M), yppasswd(3N), passwd(4), ypfiles(4).

INTERNATIONAL SUPPORT
8-bit data, messages.

NAME

yppoll - query NIS server for information about NIS map

SYNOPSIS

/usr/etc/yp/yppoll [**-h** *host*] [**-d** *domain*] *mapname*

DESCRIPTION

yppoll asks a Network Information Service (NIS) server process (see *ypserv*(1M)) to return the order number (the **time**() in seconds when the map was built – *time*(2)) and master NIS server's host name for a NIS database named *mapname*. **yppoll** then writes them to standard output. If the server uses Version 1 NIS protocol, **yppoll** uses this older protocol to communicate with it. **yppoll** also prints the old style diagnostic messages in case of failure.

See *ypfiles*(4) and *ypserv*(1M) for an overview of Network Information Service.

Options

-h *host* Ask the **ypserv** process on *host* to return the map information (see *ypserv*(1M)). If **-h** *host* is not specified, the host returned by **ypwhich** is used (see *ypwhich*(1)).

-d *domain* Use *domain* instead of the domain returned by **domainname** (see *domainname*(1)).

AUTHOR

yppoll was developed by Sun Microsystems, Inc.

SEE ALSO

domainname(1), ypwhich(1), ypserv(1M), ypfiles(4).

INTERNATIONAL SUPPORT

8-bit data, messages.

NAME

yppush - force propagation of Network Information Service database

SYNOPSIS

/usr/etc/yp/yppush [**-d** *domain*] [**-m** *maxm*] [**-t** *mint*] [**-v**] *mapname*

DESCRIPTION

yppush copies a Network Information Service (NIS) map (database), *mapname*, from the map's master NIS server to each slave NIS server. It is usually executed only on the master NIS server by shell script **ypmake** which is run either after changes are made to one or more of the master's NIS databases or when the NIS databases are first created. See *ypmake*(1M) and *ypinit*(1M) for more information on these processes.

yppush constructs a list of NIS server host names by reading the NIS map **ypservers** within the *domain*. Keys within the **ypservers** map are the host names of the machines on which the NIS servers run. **yppush** then sends a "transfer map" request to the NIS server at each host, along with the information needed by the transfer agent (the program that actually moves the map) to call back **yppush**.

When the transfer attempt is complete, whether successful or not, and the transfer agent sends **yppush** a status message, the results can be printed to standard output. Messages are printed when a transfer is not possible, such as when the request message is undeliverable or when the timeout period on responses expires.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of Network Information Service.

Options

yppush recognizes the following options:

-d *domain* Copy *mapname* to the NIS servers in *domain* rather than to the *domain* returned by **domainname** (see *domainname*(1)).

-m *maxm* Attempt to run *maxm* transfers in parallel to as many servers simultaneously. Without the **-m** option, **yppush** attempts to transfer a map to each server, one at a time. When a network has many servers, such serial transfers can result in long delays to complete all transfers. A *maxm* value greater than 1 reduces total transfer time through better utilization of CPU time at the master. *maxm* can be any value from 1 through the number of NIS servers in the domain.

-t *mint* Set the minimum timeout value to *mint* seconds. When transferring to one slave at a time, **yppush** waits up to 80 seconds for the transfer to complete, after which it begins transferring to the next slave. When multiple parallel transfers are attempted by use of the **-m** option, it may be necessary to set the transfer timeout limit to a value larger than the default 80 seconds to prevent timeouts caused by network delays related to parallel transfers.

-v Verbose mode: messages are printed when each server is called and when each response is received. If this option is omitted, only error messages are printed.

WARNINGS

In the current implementation (Version 2 NIS protocol), the transfer agent is *ypxfr*(1M) which is started by the *ypserv*(1M) program at *yppush*'s request (see *ypxfr*(1M) and *ypserv*(1M)). If *yppush* detects it is interacting with a Version 1 NIS protocol server, it uses the older protocol to send a Version 1 **YPPROC_GET** request and issues a message to that effect. Unfortunately, there is no way of knowing if or when the map transfer is performed for Version 1 servers. **yppush** prints a comment saying that a Version 1 message was sent. The system administrator should then verify by other means that the transfer actually occurred.

AUTHOR

yppush was developed by Sun Microsystems, Inc.

FILES

/usr/etc/yp/*domain*/ypservers.{dir, pag}
/usr/etc/yp/*domain/mapname*.{dir, pag}

SEE ALSO

domainname(1), ypserv(1M), ypxfr(1M), ypfiles(4).

INTERNATIONAL SUPPORT
8-bit data, messages.

NAME

ypserv, ypbind - Network Information Service server and binder processes

SYNOPSIS

/usr/etc/ypserv [**-l** *log_file*]
/etc/ypbind [**-l** *log_file*] [**-ypset**]

DESCRIPTION

The Network Information Service (NIS) provides a simple network lookup service consisting of databases and processes. The databases are files in a directory tree rooted at **/usr/etc/yp** (see *ypfiles*(4)). The processes are */usr/etc/ypserv*, the NIS database lookup server, and */etc/ypbind*, the NIS binder. Both **ypserv** and **ypbind** are daemon processes typically activated at system startup time from within */etc/netnfsrc*.

The NIS programmatic interface is described in *ypclnt*(3C). Administrative tools are described in *ypwhich*(1), *yppoll*(1M), *yppush*(1M), *ypset*(1M) and *ypxfr*(1M). Tools to see the contents of NIS maps (databases) are described in *ypcat*(1) and *ypmatch*(1). Database generation and maintenance tools are described in *makedbm*(1M), *ypinit*(1M), and *ypmake*(1M). The command to set or show the default NIS domain is *domainname*(1).

The **ypserv** daemon's primary function is to look up information in its local collection of NIS maps. It runs only on NIS server machines providing data from NIS databases. Communication to and from **ypserv** is by means of RPC. Lookup functions are described in *ypclnt*(3C) and are supplied as C-callable functions in **/lib/libc.a**.

Four lookup functions perform on a specific map within a NIS domain: **Match**, **Get_first**, **Get_next**, and **Get_all**. The **Match** operation matches a key to a record in the database and returns its associated value. The **Get_first** operation returns the first key-value pair (record) from the map, and **Get_next** enumerates (sequentially retrieves) the remainder of the records. **Get_all** returns all records in the map to the requester as the response to a single RPC request.

Two other functions supply information about the map other than normal map entries: **Get_order_number** and **Get_master_name**. The order number is the time of last modification of a map. The master name is the host name of the machine on which the master map is stored. Both order number and master name exist in the map as special key-value pairs, but the server does not return these through the normal lookup functions (if you examine the map with **makedbm** or **yppoll** – see *makedbm*(1M) or *yppoll*(1M); – however, they will be visible). Other functions are used within the NIS system and are not of general interest to NIS clients. They include **Do_you_serve_this_domain?**, **Transfer_map**, and **Reinitialize_internal_state**.

The **ypbind** daemon remembers information that lets client processes on its machine communicate with a **ypserv** process. The **ypbind** daemon must run on every machine using NIS services, both NIS servers and clients. The **ypserv** daemon may or may not be running on a NIS client machine, but it must be running somewhere on the network or be available through a gateway.

The information **ypbind** remembers is called a **binding**: the association of a NIS domain name with the internet address of the NIS server and the port on that host at which the **ypserv** process is listening for service requests. Client requests drive the binding process. As a request for an unbound domain comes in, the **ypbind** process broadcasts on the network trying to find a **ypserv** process serving maps within that NIS domain. Since the binding is established by broadcasting, at least one **ypserv** process must exist on every network. Once a binding is established for a client, it is given to subsequent client requests. Execute **ypwhich** to query the **ypbind** process (local and remote) for its current binding (see *ypwhich*(1)).

Bindings are verified before they are given to a client process. If **ypbind** is unable to transact with the **ypserv** process it is bound to, it marks the domain as unbound, tells the client process that the domain is unbound, and tries to bind again. Requests received for an unbound domain fail immediately. Generally, a bound domain is marked as unbound when the node running **ypserv** crashes or is overloaded. In such a case, **ypbind** binds to any NIS server (typically one that is less heavily loaded) available on the network.

The **ypbind** daemon also accepts requests to set its binding for a particular domain. **ypset** accesses the **Set_domain** facility; it is for unsnarling messes and is not for casual use.

Options

ypserv recognizes the following options:

`-l` *log_file* Log diagnostic and error messages to the specified *log_file*. The **ypserv** daemon writes its messages to **/usr/etc/yp/ypserv.log** if **ypserv** is started without the **-l** (ell) option and the file exists. The **ypbind** daemon writes its messages directly to the system console, **/dev/console**, if **ypbind** is started without the **-l** (ell) option.

-ypset Allow **ypset** to be used to change the binding (see *ypset*(1M)). For maximum security this option should be used only when debugging the network from a remote machine.

Information logged to the file includes date and time of the message, the host name, process id and name of the function generating the message, and the message itself. Note that different services can share a single log file since enough information is included to uniquely identify each message.

AUTHOR

ypserv was developed by Sun Microsystems, Inc.

FILES

/usr/etc/yp/ypserv.log
default ypserv error log file

SEE ALSO

domainname(1), ypcat(1), ypmatch(1), yppasswd(1), ypwhich(1), makedbm(1M), rpcinfo(1M), ypinit(1M), ypmake(1M), yppasswdd(1M), yppoll(1M), yppush(1M), ypset(1M), ypxfr(1M), ypclnt(3C), yppasswd(3N), ypfiles(4).

INTERNATIONAL SUPPORT

messages

NAME

ypset - bind to particular Network Information Service server

SYNOPSIS

/usr/etc/yp/ypset [**-V1** | **-V2**] [**-h** *host*] [**-d** *domain*] *server*

DESCRIPTION

ypset tells **ypbind** to get Network Information Service (NIS) services for the specified *domain* from the **ypserv** process running on *server* (see *ypserv*(1M) and *ypbind*(1M)). *server* is the NIS server that the NIS client binds to, and is specified as either a host name or an IP address. If *server* is down or is not running **ypserv**, this is not discovered until a local NIS client process tries to obtain a binding for the domain. The **ypbind** daemon then tests the binding set by **ypset**. If the binding cannot be made to the requested server, **ypbind** attempts to rebind to another server in the same domain.

The **ypset** command is useful for binding a client node that is not on a broadcast network, since broadcasting is the method by which **ypbind** locates a NIS server. If a client node exists on a broadcast network which has no NIS server running, and if there is a network with one running that is available via a gateway, **ypset** can establish a binding through that gateway. It is also useful for debugging NIS client applications such as when a NIS map exists only at a single NIS server.

In cases where several hosts on the local net are supplying NIS services, it is possible for **ypbind** to rebind to another host, even while you attempt to find out if the **ypset** operation succeeded. For example, typing **ypset host1** followed by **ypwhich** and receiving the reply **host2** may be confusing. It could occur when *host1* does not respond to **ypbind** because its **ypserv** process is not running or is overloaded, and *host2*, running **ypserv**, gets the binding.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of the Network Information Service.

Options

ypset recognizes the following options and comman-line arguments:

-V1	Bind *server* for the (old) Version 1 NIS protocol.
-V2	Bind *server* for the (current) Version 2 NIS protocol. If neither version is supplied, **ypset** first attempts to set the binding for the Version 2 protocol. If this fails, **ypset** then attempts to set the binding for the Version 1 protocol.
-h *host*	Set the binding on *host* instead of locally. *host* can be specified as a host name or an IP address.
-d *domain*	Use *domain* instead of the default domain returned by **domainname** (see *domainname*(1)).

DIAGNOTICS

Sorry, ypbind on host 'name' has rejected your request.
The user is not root, or ypbind was run without the **-ypset** flags. See **ypserv(1m)** for explanations of the **-ypset** flags.

Sorry, I couldn't send my rpc message to ypbind on host 'name'.
The user is not root, or ypbind was run without one of the **-ypset** flags. See **ypserv(1m)** for explanations of the **-ypset** flags.

WARNINGS

The *server* is the NIS server to bind to, specified as either a host name or an IP address. If *server* is a host name, **ypset** uses the NIS services' **hosts** database (built from **/etc/hosts** on the master server) to resolve the name to an IP address. This process works only if the node currently has a valid binding for the domain in question. In most cases, *server* should be specified as an IP address.

AUTHOR

ypset was developed by Sun Microsystems, Inc.

SEE ALSO

domainname(1), ypwhich(1), ypserv(1m), ypfiles(4).

NAME

ypxfr, ypxfr_1perday, ypxfr_1perhour, ypxfr_2perday - transfer NIS database from server to local node

SYNOPSIS

/usr/etc/yp/ypxfr [**-h** *host*] [**-f**] [**-d** *domain*] [**-c**] [**-C** *tid prog ipaddr port*] *mapname*

DESCRIPTION

ypxfr copies a Network Information Service (NIS) map (database) to the local host from a NIS server by using the NIS services. A map can be copied regardless of its age, or it can be copied depending on whether its modification time (order number) is more recent than that of the local map.

The **ypxfr** command creates a temporary map in directory **/usr/etc/yp/domain** where *domain* is the NIS *domain*. The **ypxfr** command fills the map with *mapname* entries, obtains the map parameters (master and order number), and loads them. It then clears the old version of *mapname* and moves the temporary map to the existing *mapname*.

If **ypxfr** is run interactively, it writes messages to standard output. If **ypxfr** is invoked without a controlling terminal and if the log file **/usr/etc/yp/ypxfr.log** exists, **ypxfr** appends all its messages to that file. Since **ypxfr** is usually run from root's **crontab** file (see *crontab*(1)) or by **yppush** (see *yppush*(1M)), the log file can retain a record of what **ypxfr** attempted and what the results were.

To maintain consistency between NIS servers, **ypxfr** should be executed periodically for every map in the NIS. Different maps change at different rates. For example, the **services.byname** map may not change for months at a time, and might therefore be checked for changes only once a day, such as in the early morning hours. However, **passwd.byname** may change several times per day, so hourly checks for updates might be more appropriate.

A **crontab** file can perform these periodic checks and transfers automatically. Rather than having a separate **crontab** file for each map, **ypxfr** requests can be grouped in a shell script to update several maps at once. Example scripts (mnemonically named) are in **/usr/etc/yp**: **ypxfr_1perday**, **ypxfr_2perday**, and **ypxfr_1perhour**. They serve as reasonable rough drafts that can be changed as appropriate.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of the Network Information Service.

Options

ypxfr recognizes the following options and command-line arguments:

-h *host* Obtain the map from *host*, regardless of its master server. If this option is not used, **ypxfr** asks the NIS service for the master's host name and tries to obtain its map. The *host* can be a name or an IP address of the form *a.b.c.d*.

-f Force the map to be copied, even if its order number at the remote NIS server is not more recent than the order number of the local map.

-d *domain* Copy the map from a NIS server in *domain* rather than the *domain* returned by **domainname** (see *domainname*(1)).

-c Do not send a "clear current map" request to the local **ypserv** process. Use this flag if **ypserv** is not running locally when you are running **ypxfr**. Otherwise, **ypxfr** complains that it cannot talk to the local **ypserv**, and the transfer fails. If **ypserv** is running locally, do not use this flag.

-C *tid prog ipaddr port*
This option is used by **ypserv** *only*. When **ypserv** invokes **ypxfr**, it specifies that **ypxfr** should call back a **yppush** process (that initiated the transfer) at the host with IP address *ipaddr*, registered as program number *prog*, listening on port *port*, and waiting for a response to transaction *tid*.

AUTHOR

Sun Microsystems, Inc.

FILES

/usr/etc/yp/ypxfr.log log file

The following scripts are suggested for use with **cron**.

`/usr/etc/yp/ypxfr_1perday`	run one transfer per day
`/usr/etc/yp/ypxfr_2perday`	run two transfers per day
`/usr/etc/yp/ypxfr_1perhour`	hourly transfers of "volatile" maps

SEE ALSO

crontab(1), domainname(1), cron(1M), ypinit(1M), yppush(1M), ypserv(1M), ypfiles(4).

Section 4:
File Formats

NAME

intro - introduction to file formats

DESCRIPTION

This section outlines the formats of various files. The C `struct` declarations for the file formats are given where applicable. Usually, these structures can be found in directories **/usr/include** or **/usr/include/sys**.

SEE ALSO

hier(5).

The introduction to this manual.

NAME

a.out - assembler and link editor output

REMARKS

A separate manual entry describes each implementation of the **a.out** file format for Series 300/400 and Series 700/800 systems.

DESCRIPTION

The **a.out** (i.e., object file) format is completely machine-dependent except for the first word, which contains a magic number as defined in *magic*(4).

The archive symbol table format is also completely machine-dependent except for its name in the archive. See *ar*(4) for a description of the format of the archive symbol table.

SEE ALSO

crt0(3), end(3C), a.out_300(4), a.out_800(4), ar(4), magic(4).

NAME

a.out - assembler and link editor output

Remarks:

This manual entry describes the `a.out` file format for Series 300/400 computers. Refer to other *a.out**(4) manual entries for descriptions of other implementations.

DESCRIPTION

The `a.out` file is the output file of the link editor `ld` (see *ld*(1). The linker makes `a.out` executable if there were no linking errors and no unresolved external references. The assembler *as* (or `ld` with the `-r` option) produces non-executable files with the same basic structure.

File `a.out` has eight defined sections: a header, the program text and data segments, a Pascal interface section, a symbol table, a supplementary symbol table, and text and data relocation information (in that order). Pascal interface text will only be present in those Pascal code segments that have not been linked. The symbol table may be missing if the program was linked with the `ld -s` option, or if the symbol table and debug information were removed by `strip` (see *strip*(1)). The supplementary symbol table is present only in files containing position-independent code, or in files that were produced with the Pascal compiler or the `+s` option to the assembler. Also note that relocation information is not normally present in executable files.

In addition to these sections, there may be one or more extensions. Each extension is preceded by a header consisting of an extension-independent part and an extension-dependent part. Currently defined extensions include one for dynamic loading support and one for debugger support. The dynamic loader extension is actually placed within the text segment of shared library files (created with the `-b` option to `ld`) and executables that use shared libraries. The debugger extension is placed after the relocation sections. HP-UX compilers create this information under control of the `-g` option.

When an `a.out` file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. The text segment begins at location 0x0 in the core image; the header is not loaded. If the magic number (the first field in the header) is `EXEC_MAGIC`, it indicates that the text segment is not to be write-protected or shared, so the data segment will be contiguous with the text segment. If the magic number is `SHARE_MAGIC` or `DEMAND_MAGIC`, the data segment begins at the first 0 mod 0x1000 byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same `a.out` file, they will share a single text segment. If the magic number is `DEMAND_MAGIC`, the text and data segments are not read in from the file until they are referenced by the program.

The stack will occupy the highest possible locations in the core image and grow downward (the stack is automatically extended as required). The data segment is only extended as requested by the `brk()` system call (see *brk*(2)).

Shared libraries, indicated by the magic number `SHL_MAGIC`, are similar to demand-loaded executables, except that they are loaded by the dynamic loader **/lib/dld.sl** (see *dld.sl*(5)) at some point during startup, rather than by `exec()`, and that the load address is arbitrary. No stack segment is set up for shared libraries; they share the stack used by the executable. The data and bss segments of a shared library are shared on a page by page basis by all processes using the library. Whenever a process writes to a shared library data or bss segment, a modified copy of that page is made for the process. Dynamic load libraries, indicated by the magic number `DL_MAGIC`, are analagous to shared libraries except that the text segment is also made copy on write in order to support dynamic relocation.

The start of the text segment in the `a.out` file is given by the macro `TEXT_OFFSET(`*hdr*`)`, where *hdr* is a copy of the file header. The macro `DATA_OFFSET(`*hdr*`)` provides the starting location of the data segment.

The value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation information (discussed below) for that word, then the value of the word as stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

Header

The format of the `a.out` header for the MC68000 is as follows (segment sizes are in bytes):

```
struct exec {
    MAGIC   a_magic;        /* magic number */
    short   a_stamp;        /* version id */
    short   a_highwater;    /* shlib highwater mark */
    long    a_miscinfo;     /* miscellaneous info */
    long    a_text;         /* size of text segment */
    long    a_data;         /* size of data segment */
    long    a_bss;          /* size of bss segment */
    long    a_trsize;       /* text relocation size */
    long    a_drsize;       /* data relocation size */
    long    a_pasint;       /* Pascal interface size */
    long    a_lesyms;       /* symbol table size */
    long    a_spared;
    long    a_entry;        /* entry point */
    long    a_spares;
    long    a_supsym;       /* supplementary symtab size */
    long    a_drelocs;      /* nonpic relocations */
    long    a_extension;    /* file offset of extension */
};
```

Pascal Interface Section

The Pascal interface section consists of the ASCII representation of the interface text for that Pascal module.

The start of the Pascal interface section is given by the macro `MODCAL_OFFSET(`*hdr*`)`.

Symbol Table

The symbol table consists of entries of the form:

```
struct nlist {
    long            n_value;
    unsigned char    n_type;
    unsigned char    n_length;
    short           n_almod;
    int             n_dlt:1;
    int             n_plt:1;
    int             n_dreloc:1;
    int             n_list:1;
    int             n_unused:12;
};
```

This structure is followed by `n_length` ASCII characters which compose the symbol name.

The `n_type` field indicates the type of the symbol; the following values are possible:

UNDEF	undefined symbol
ABS	absolute symbol
TEXT	text segment symbol
DATA	data segment symbol
BSS	bss segment symbol

One of these values ORed with 040 indicates an external symbol. One of these values ORed with 020 indicates an aligned symbol. One of these values ORed with 0100 indicates a secondary definition.

The bit fields are used by the linker for the relocation of position-independent code (generated with the `+z` option to the compilers) when creating a shared library.

The start of the symbol table is given by the macro `LESYM_OFFSET(`*hdr*`)`.

In object files that were compiled as position-independent code, generated by the Pascal compiler, or assembled with the `as +s` option, a supplementary symbol table follows the standard symbol table. This table is a simple array of eight-byte structures which stand in one-to-one correspondence with the symbol table entries. The fields of this structure are used by the linker when creating a shared library.

Relocation

If relocation information is present, it amounts to eight bytes per relocatable datum.

The format of the relocation data is:

```
struct r_info   {
    long   r_address;
    short  r_symbolnum;
    char   r_segment;
    char   r_length;
};
```

The `r_address` field indicates the position of the relocation within the segment.

The `r_segment` field indicates the segment referred to by the text or data word associated with the relocation word:

`RTEXT`	indicates the reference is to the text segment;
`RDATA`	indicates the reference is to initialized data;
`RBSS`	indicates the reference is to bss (uninitialized data);
`REXT`	indicates the reference is to an undefined external symbol;
`RPC`	indicates the reference is PC relative;
`RDLT`	indicates the reference is to an offset from the base of the data linkage table;
`RPLT`	indicates the reference is to an entry within the procedure linkage table.

The latter three relocation types are found primarily in position-independent code.

The `r_symbolnum` field contains a symbol number in the case of external references, and is unused otherwise. The first symbol is numbered 0, the second 1, etc.

The `r_length` field indicates the length of the datum to be relocated.

`RBYTE`	indicates it is a byte
`RWORD`	indicates it is a short
`RLONG`	indicates it is a long
`RALIGN`	indicates it is a special align symbol

The start of the text relocation section is provided by the macro `RTEXT_OFFSET` (*hdr*) .

The start of the data relocation section is provided by the macro `RDATA_OFFSET` (*hdr*) .

Extensions

The `a_extension` field of the header gives the file offset of the first extension, and each extension header gives the file offset of the next. The dynamic loader extension, if present, will be the first extension present, and the debugger extension, if present, will be next. Additional extensions can be defined by certain languages or tools. When present, they are placed after the dynamic loader and debugger extensions.

The extension header is of the form

```
struct header_extension
{
    union
    {
        long spare1[13];
        struct _dl_header dl_header;
        struct _debug_header debug_header;
    } e_spec;
    short e_header;
    short e_version;
    long e_size;
    long e_extension;
};
```

The union allows for extension specific information to be included in the header. The `e_header` field contains a constant which identifies the extension type. Currently defined values include `DL_HEADER` and `DEBUG_HEADER`. The `e_version` field can be used to identify different versions of an extension. The `e_size` field gives the collective size for the extension; `e_extension` gives the file offset of the next

extension.

For more information about the dynamic loader extension and the debug extension, consult the files `/usr/include/shl.h` and `/usr/include/debug.h`, respectively.

SEE ALSO

as_300(1), ld(1), nm_300(1), strip(1), crt0(3), end(3C), a.out_800(4), magic(4).

NAME

a.out - assembler and link editor output

SYNOPSIS

```
#include <a.out.h>
```

Remarks:

This manual entry describes the `a.out` file format for Series 700 and Series 800 computers. Refer to other *a.out*(4) manual entries for descriptions of other valid implementations.

DESCRIPTION

The file name `a.out` is the output file from the assembler (see *as*(1)), compilers, and the linker (see *ld*(1)). The assembler and compilers create relocatable object files ready for input to the linker; the linker creates executable object files and shared library files.

An object file consists of a file header, auxiliary headers, space dictionary, subspace dictionary, symbol table, relocation information, compiler records, space string table, symbol string table, and the data for initialized code and data. Not all of these sections are required for all object files. The file must begin with the file header, but the remaining sections do not have to be in any particular order; the file header contains pointers to each of the other sections of the file.

A relocatable object file, created by the assembler or compiler, must contain at least the following sections: file header, space dictionary, subspace dictionary, symbol table, relocation information, space string table, symbol string table, and code and data. It may also contain auxiliary headers and compiler records. Relocatable files generally contain unresolved symbols; the linker combines relocatable files and searches libraries to produce an executable file. The linker can also be used to combine relocatable files and produce a new relocatable file as output, suitable for input to a subsequent linker run.

An executable file, created by the linker, typically contains the following sections: file header, an HP-UX auxiliary header, space dictionary, subspace dictionary, symbol table, space string table, symbol string table, and code and data. The linker also copies any auxiliary headers and compiler records from the input files to the output file. If the file has been stripped (see *strip*(1)), it will not contain a symbol table, symbol string table, or compiler records. An executable file must not contain any unresolved symbols.

A shared library file, created by the linker, contains the same sections found in an executable file, with additional information added to the code section of the file. This additional information contains a header, export table, import table, and dynamic relocation records to be used by the dynamic loader.

Programs for the Series 700/800 architecture consist of two loadable spaces: a shared, non-writable, code space named `$TEXT$`; and a private, writable, data space named `$PRIVATE$`. A program may contain other non-loadable spaces that contain data needed by development tools; for example, symbolic debugging information is contained in a space named `$DEBUG$`. The linker treats loadable and unloadable spaces exactly the same, so the full generality of symbol resolution and relocation is available for the symbolic debugging information. Spaces have an addressing range of 4,294,967,296 (2^32) bytes; each loadable space is divided into four 1,073,741,824 (2^30) byte quadrants. The HP-UX operating system places all code in the first quadrant of the `$TEXT$` space, all data in the second quadrant of the `$PRIVATE$` space, and all shared library code into the third quadrant of shared memory space.

Each space is also divided into logical units called subspaces. When the linker combines relocatable object files, it groups all subspaces from the input files by name, then arranges the groups within the space by a sort key associated with each subspace. Subspaces are not architecturally significant; they merely provide a mechanism for combining individual parts of spaces independently from many input files. Some typical subspaces in a program are shown in the following table:

`$SHLIB_INFO$`	Information needed for dynamic loading.
`$MILLICODE$`	Code for millicode routines
`$LIT$`	Sharable literals
`$CODE$`	Code
`$UNWIND$`	Stack unwind information
`$GLOBAL$`	Outer block declarations for Pascal
`$DATA$`	Static initialized data
`$COMMON$`	FORTRAN common
`$SHLIB_DATA$`	Imported data from referenced shared libraries.
`$BSS$`	Uninitialized data

Subspaces can be initialized or uninitialized (although typically, only `$BSS$` is uninitialized). The subspace dictionary entry for an initialized subspace contains a file pointer to the initialization data, while the entry for an uninitialized subspace contains only a 32-bit pattern used to initialize the entire area at load time.

In a relocatable file, initialized code and data often contains references to locations elsewhere in the file, and to unresolved symbols defined in other files. These references are patched at link time using the relocation information. Each entry in the relocation information (a "fixup") specifies a location within the initialized data for a subspace, and an expression that defines the actual value that should be placed at that location, relative to one or two symbols.

The linker summarizes the subspace dictionary in the HP-UX auxiliary header when creating an executable file. HP-UX programs contain only three separate sections: one for the code, one for initialized data, and one for uninitialized data. By convention, this auxiliary header is placed immediately following the file header.

When an `a.out` file is loaded into memory for execution, three areas of memory are set up: the a.out code is loaded into the first quadrant of a new, sharable space; the data (initialized followed by uninitialized) is loaded into the second quadrant of a new, private space; and a stack is created beginning at a fixed address near the middle of the second quadrant of the data space.

If the `a.out` file uses shared libraries then the dynamic loader `/lib/dld.sl` is loaded into memory, and called to map all shared libraries, requested by the program, into memory. The shared library text is loaded into the third quadrant of the shared memory space, and the shared library data is allocated in the second quadrant of the data space.

The file format described here is a common format for all operating systems designed for HP's Precision Architecture. Therefore, there are some fields and structures that are not used on HP-UX or have been reserved for future use.

File Header

The format of the file header is described by the following structure declaration from <`filehdr.h`>.

```
struct header {
    short int system_id;                    /* system id */
    short int a_magic;                      /* magic number */
    unsigned int version_id;                /* a.out format version */
    struct sys_clock file_time;             /* timestamp */
    unsigned int entry_space;               /* reserved */
    unsigned int entry_subspace;            /* reserved */
    unsigned int entry_offset;              /* reserved */
    unsigned int aux_header_location;       /* file ptr to aux hdrs */
    unsigned int aux_header_size;           /* sizeof aux hdrs */
    unsigned int som_length;                /* length of object module */
    unsigned int presumed_dp;               /* reserved */
    unsigned int space_location;            /* file ptr to space dict */
    unsigned int space_total;               /* # of spaces */
    unsigned int subspace_location;         /* file ptr to subsp dict */
    unsigned int subspace_total;            /* # of subspaces */
    unsigned int loader_fixup_location;     /* reserved */
    unsigned int loader_fixup_total;        /* reserved */
    unsigned int space_strings_location;    /* file ptr to sp. strings */
    unsigned int space_strings_size;        /* sizeof sp. strings */
    unsigned int init_array_location;       /* reserved */
    unsigned int init_array_total;          /* reserved */
    unsigned int compiler_location;         /* file ptr to comp recs */
    unsigned int compiler_total;            /* # of compiler recs */
    unsigned int symbol_location;           /* file ptr to sym table */
    unsigned int symbol_total;              /* # of symbols */
    unsigned int fixup_request_location;    /* file ptr to fixups */
    unsigned int fixup_request_total;       /* # of fixups */
    unsigned int symbol_strings_location;   /* file ptr to sym strings */
    unsigned int symbol_strings_size;       /* sizeof sym strings */
```

```
    unsigned int unloadable_sp_location;  /* file ptr to debug info */
    unsigned int unloadable_sp_size;      /* size of debug info */
    unsigned int checksum;                /* header checksum */
};
```

The timestamp is a two-word structure as shown below. If unused, both fields are zero.

```
struct sys_clock {
    unsigned int secs;
    unsigned int nanosecs;
};
```

Auxiliary Headers

The auxiliary headers are contained in a single contiguous area in the file, and are located by a pointer in the file header. Auxiliary headers are used for two purposes: users can attach version and copyright strings to an object file, and an auxiliary header contains the information needed to load an executable program. In an executable program, the HP-UX auxiliary header must precede all other auxiliary headers. The following declarations are found in <aouthdr.h>.

```
struct aux_id {
    unsigned int  mandatory : 1;  /* reserved */
    unsigned int  copy : 1;       /* reserved */
    unsigned int  append : 1;     /* reserved */
    unsigned int  ignore : 1;     /* reserved */
    unsigned int  reserved : 12;  /* reserved */
    unsigned int  type : 16;      /* aux hdr type */
    unsigned int  length;         /* sizeof rest of aux hdr */
};

/* Values for the aux_id.type field */
#define HPUX_AUX_ID 4
#define VERSION_AUX_ID 6
#define COPYRIGHT_AUX_ID 9
#define SHLIB_VERSION_AUX_ID 10

struct som_exec_auxhdr {          /* HP-UX auxiliary header */
    struct   aux_id som_auxhdr;   /* aux header id */
    long           exec_tsize;    /* text size */
    long           exec_tmem;     /* start address of text */
    long           exec_tfile;    /* file ptr to text */
    long           exec_dsize;    /* data size */
    long           exec_dmem;     /* start address of data */
    long           exec_dfile;    /* file ptr to data */
    long           exec_bsize;    /* bss size */
    long           exec_entry;    /* address of entry point */
    long           exec_flags;    /* loader flags */
    long           exec_bfill;    /* bss initialization value */
};

/* Values for exec_flags */
#define TRAP_NIL_PTRS    01

struct user_string_aux_hdr {        /* Version string auxiliary header */
    struct aux_id  header_id;       /* aux header id */
    unsigned int   string_length;   /* strlen(user_string) */
    char           user_string[1];  /* user-defined string */
};

struct copyright_aux_hdr {          /* Copyright string auxiliary header */
    struct aux_id  header_id;       /* aux header id */
    unsigned int   string_length;   /* strlen(user_string) */
    char           copyright[1];    /* user-defined string */
};
struct shlib_version_aux_hdr {
```

```
    struct aux_id  header_id;       /* aux header id */
    short  version;                 /* version number */
};
```

Space Dictionary

The space dictionary consists of a sequence of space records as defined in <spacehdr.h>.

```
struct space_dictionary_record {"
    union name_pt name;             /* index to space name */
    unsigned int  is_loadable: 1;   /* space is loadable */
    unsigned int  is_defined: 1;    /* space is defined within file */
    unsigned int  is_private: 1;    /* space is not sharable */
    unsigned int  reserved: 13;     /* reserved */
    unsigned int  sort_key: 8;      /* sort key for space */
    unsigned int  reserved2: 8;     /* reserved */
    int           space_number;     /* space index */
    int           subspace_index;   /* index to first subspace */
    unsigned int  subspace_quantity; /* # of subspaces in space */
    int           loader_fix_index;     /* reserved */
    unsigned int  loader_fix_quantity;  /* reserved */
    int           init_pointer_index;   /* reserved */
    unsigned int  init_pointer_quantity; /* reserved */
};
```

The strings for the space names are contained in the space strings table, which is located by a pointer in the file header. Each entry in the space strings table is preceded by a 4-byte integer that defines the length of the string, and is terminated by one to five null characters to pad the string out to a word boundary. Indices to this table are relative to the start of the table, and point to the first byte of the string (not the preceding length word). The union defined below is used for all such string pointers; the character pointer is defined for programs that read the string table into memory and wish to relocate in-memory copies of space records.

```
union name_pt {
    char          *n_name;
    unsigned int   n_strx;
};
```

Subspace Dictionary

The subspace dictionary consists of a sequence of subspace records as defined in <scnhdr.h>. Strings for subspace names are contained in the space strings table.

```
struct subspace_dictionary_record {
    int           space_index;
    unsigned int  access_control_bits: 7; /* reserved */
    unsigned int  memory_resident: 1;     /* reserved */
    unsigned int  dup_common: 1;          /* COBOL-style common */
    unsigned int  is_common: 1;           /* subspace is a common block */
    unsigned int  is_loadable: 1;         /* subspace is loadable */
    unsigned int  quadrant: 2;            /* reserved */
    unsigned int  initially_frozen: 1;    /* reserved */
    unsigned int  is_first: 1;            /* reserved */
    unsigned int  code_only: 1;           /* subspace contains only code */
    unsigned int  sort_key: 8;            /* subspace sort key */
    unsigned int  replicate_init: 1;      /* reserved */
    unsigned int  continuation: 1;        /* reserved */
    unsigned int  reserved: 6;            /* reserved */
    int           file_loc_init_value;    /* file location or init value */
    unsigned int  initialization_length;  /* length of initialization */
    unsigned int  subspace_start;         /* starting offset */
    unsigned int  subspace_length;        /* total subspace length */
    unsigned int  reserved2: 16;          /* reserved */
    unsigned int  alignment: 16;          /* alignment required */
```

```
    union name_pt name;                    /* index of subspace name */
    int           fixup_request_index;     /* index to first fixup */
    unsigned int  fixup_request_quantity; /* # of fixup requests */
};
```

Symbol Table

The symbol table consists of a sequence of entries described by the structure shown below, from <syms.h>. Strings for symbol and qualifier names are contained in the symbol strings table, whose structure is identical with the space strings table.

```
struct symbol_dictionary_record {
    unsigned int    hidden: 1;           /* reserved */
    unsigned int    symbol_type: 7;      /* symbol type */
    unsigned int    symbol_scope: 4;     /* symbol value */
    unsigned int    check_level: 3;      /* type checking level */
    unsigned int    must_qualify: 1;     /* qualifier required */
    unsigned int    initially_frozen: 1; /* reserved */
    unsigned int    memory_resident: 1;  /* reserved */
    unsigned int    is_common: 1;        /* common block */
    unsigned int    dup_common: 1;       /* COBOL-style common */
    unsigned int    xleast: 2;           /* reserved */
    unsigned int    arg_reloc: 10;       /* parameter relocation bits */
    union name_pt   name;                /* index to symbol name */
    union name_pt   qualifier_name;      /* index to qual name */
    unsigned int    symbol_info;         /* subspace index */
    unsigned int    symbol_value;        /* symbol value */
};

/* Values for symbol_type */
#define ST_NULL          0    /* unused symbol entry */
#define ST_ABSOLUTE      1    /* non-relocatable symbol */
#define ST_DATA          2    /* data symbol */
#define ST_CODE          3    /* generic code symbol */
#define ST_PRI_PROG      4    /* program entry point */
#define ST_SEC_PROG      5    /* secondary prog entry point*/
#define ST_ENTRY         6    /* procedure entry point */
#define ST_STORAGE       7    /* storage request */
#define ST_STUB          8    /* reserved */
#define ST_MODULE        9    /* Pascal module name */
#define ST_SYM_EXT       10   /* symbol extension record */
#define ST_ARG_EXT       11   /* argument extension record */
#define ST_MILLICODE     12   /* millicode entry point */
#define ST_PLABEL        13   /* reserved */
#define ST_OCT_DIS       14   /* reserved */
#define ST_MILLI_EXT     15   /* reserved */

/* Values for symbol_scope */
#define SS_UNSAT         0    /* unsatisfied reference */
#define SS_EXTERNAL      1    /* reserved */
#define SS_LOCAL         2    /* local symbol */
#define SS_UNIVERSAL     3    /* global symbol */
```

The meaning of the symbol value depends on the symbol type. For the code symbols (generic code, program entry points, procedure and millicode entry points), the low-order two bits of the symbol value encode the execution privilege level, which is not used on HP-UX, but is generally set to 3. The symbol value with those bits masked out is the address of the symbol (which is always a multiple of 4). For data symbols, the symbol value is simply the address of the symbol. For storage requests, the symbol value is the number of bytes requested; the linker allocates space for the largest request for each symbol in the BSS subspaces, unless a local or universal symbol is found for that symbol (in which case the storage request is treated like an unsatisfied reference).

If a relocatable file is compiled with parameter type checking, extension records follow symbols that define and reference procedure entry points and global variables. The first extension record, the *symbol extension record*, defines the type of the return value or global variable, and (if a procedure or function) the number of parameters and the types of the first three parameters. If more parameter type descriptors are needed, one or more *argument extension records* follow, each containing four more descriptors. A check level of 0 specifies no type checking; no extension records follow. A check level of 1 or more specifies checking of the return value or global variable type. A check level of 2 or more specifies checking of the number of parameters, and a check level of 3 specifies checking the types of each individual parameter. The linker performs the requested level of type checking between unsatisfied symbols and local or universal symbols as it resolves symbol references.

```
union arg_descriptor {
    struct {
    unsigned int    reserved: 3;    /* not used */
    unsigned int    packing: 1;     /* reserved */
    unsigned int    alignment: 4;   /* byte alignment */
    unsigned int    reserved2: 1;   /* not used */
    unsigned int    mode: 3;        /* use of symbol */
    unsigned int    structure: 4;   /* structure of symbol */
    unsigned int    hash: 1;        /* set if arg_type is hashed */
    int             arg_type: 15;   /* data type */
    }               arg_desc;
    unsigned int    word;
};

struct symbol_extension_record {
    unsigned int    type:          8;          /* always ST_SYM_EXT */
    unsigned int    max_num_args: 8;           /* max # of parameters */
    unsigned int    min_num_args: 8;           /* min # of parameters */
    unsigned int    num_args:      8;          /* actual # of parameters */
    union arg_descriptor symbol_desc;          /* symbol type desc. */
    union arg_descriptor argument_desc[3];     /* first 3 parameters */
};

struct argument_desc_array {
    unsigned int    type:       8;             /* always ST_ARG_EXT */
    unsigned int    reserved: 24;              /* not used */
    union arg_descriptor argument_desc[4];     /* next 4 parameters */
};
```

The values for the *alignment*, *mode*, *structure*, and *arg_type* (when the data type is not hashed) fields are given in the following table.

value	alignment	mode	structure	arg_type
0	byte	any	any	any
1	half-word	value parm	scalar	void
2	word	reference parm	array	signed byte
3	dbl word	value-result	struct	unsigned byte
4		name	pointer	signed short
5		variable	long ptr	unsigned short
6	64-byte	function return	C string	signed long
7		procedure	Pascal string	unsigned long
8		long ref parm	procedure	signed dbl word
9			function	unsigned dbl word
10			label	short real
11	page			real
12				long real
13				short complex
14				complex
15				long complex
16				packed decimal
17				struct/array

For procedure entry points, the parameter relocation bits define the locations of the formal parameters and the return value. Normally, the first four words of the parameter list are passed in general registers (`r26-r23`) instead of on the stack, and the return value is returned in `r29`. Floating-point parameters in this range are passed instead in floating-point registers (`fr4-fr7`) and a floating-point value is returned in `fr4`. The parameter relocation bits consist of five pairs of bits that describe the first four words of the parameter list and the return value. The leftmost pair of bits describes the first parameter word, and the rightmost pair of bits describes the return value. The meanings of these bits are shown in the following table.

bits	meaning
00	no parameter or return value
01	parameter or return value in general register
10	parameter or return value in floating-point register
11	double-precision floating-point value

For double-precision floating-point parameters, the odd-numbered parameter word should be marked `11` and the even-numbered parameter word should be marked `10`. Double-precision return values are simply marked `11`.

Every procedure call is tagged with a similar set of bits (see Relocation Information, below), so that the linker can match each call with the expectations of the procedure entry point. If the call and entry point mismatch, the linker creates a stub that relocates the parameters and return value as appropriate.

Relocation Information

Each initialized subspace defines a range of fixups that apply to the data in that subspace. A fixup request is associated with every word that requires relocation or that contains a reference to an unsatisfied symbol. In relocatable object files created prior to HP-UX Release 3.0 on Series 800 systems each fixup request is a five-word structure describing a code or data word to be patched at link time. Object files created on Release 3.0 or later contain variable-length fixup requests that describe every byte of the subspace. The *version_id* field in the file header distinguishes these two formats; the constant `VERSION_ID` is found in older object files, and the constant `NEW_VERSION_ID` is found in newer ones.

In older object files, fixups can compute an expression involving zero, one, or two symbols and a constant, then extract a field of bits from that result and deposit those bits in any of several different formats (corresponding to the Precision Architecture instruction set). The *fixup_request_index* field in the subspace dictionary entry indexes into the fixup request area defined by the file header, and the *fixup_request_quantity* field refers to the number of fixup requests used for that subspace. The structure of a fixup request is contained in `<reloc.h>`.

```
struct fixup_request_record {
    unsigned int    need_data_ref:   1;     /* reserved */
    unsigned int    arg_reloc:      10;     /* parameter relocation bits */
```

```
    unsigned int    expression_type: 5;     /* how to compute value */
    unsigned int    exec_level:      2;     /* reserved */
    unsigned int    fixup_format:    6;     /* how to deposit bits */
    unsigned int    fixup_field:     8;     /* field to extract */
    unsigned int    subspace_offset;        /* subspace offset of word */
    unsigned int    symbol_index_one;       /* index of first symbol */
    unsigned int    symbol_index_two;       /* index of second symbol */
    int             fixup_constant;         /* constant */
};

/* Values for expression_type */
#define e_one       0  /* symbol1 + constant */
#define e_two       1  /* symbol1 - symbol2 + constant */
#define e_pcrel     2  /* symbol1 - pc + constant */
#define e_con       3  /* constant */
#define e_plabel    7  /* symbol1 + constant */
#define e_abs      18  /* reserved */

/* Values for fixup_field (assembler mnemonics shown) */
#define e_fsel      0  /* F': no change */
#define e_lssel     1  /* LS': inverse of RS' */
#define e_rssel     2  /* RS': rightmost 11 bits, signed */
#define e_lsel      3  /* L': leftmost 21 bits */
#define e_rsel      4  /* R': rightmost 11 bits */
#define e_ldsel     5  /* LD': inverse of RD' */
#define e_rdsel     6  /* RD': rightmost 11 bits, filled left with ones */
#define e_lrsel     7  /* LR': L' with  rounded" constant */"
#define e_rrsel     8  /* RR': R' with  rounded" constant */"

/* Values for fixup_format (typical instructions shown) */"
#define i_exp14     0  /* 14-bit immediate (LDW, STW) */
#define i_exp21     1  /* 21-bit immediate (LDIL, ADDIL) */
#define i_exp11     2  /* 11-bit immediate (ADDI, SUBI) */
#define i_rel17     3  /* 17-bit pc-relative (BL) */
#define i_rel12     4  /* 12 bit pc-relative (COMBT, COMBF, etc.) */
#define i_data      5  /* whole word */
#define i_none      6  /* not used */
#define i_abs17     7  /* 17-bit absolute (BE, BLE) */
#define i_milli     8  /* 17-bit millicode call (BLE) */
#define i_break     9  /* reserved (no effect on HP-UX) */
```

In newer object files, relocation entries consist of a stream of bytes. The *fixup_request_index* field in the subspace dictionary entry is a byte offset into the fixup dictionary defined by the file header, and the *fixup_request_quantity* field defines the length of the fixup request stream, in bytes, for that subspace. The first byte of each fixup request (the opcode) identifies the request and determines the length of the request.

In general, the fixup stream is a series of linker instructions that governs how the linker places data in the `a.out` file. Certain fixup requests cause the linker to copy one or more bytes from the input subspace to the output subspace without change, while others direct the linker to relocate words or resolve external references. Still others direct the linker to insert zeroes in the output subspace or to leave areas uninitialized without copying any data from the input subspace, and others describe points in the code without contributing any new data to the output file.

The include file <`reloc.h`> defines constants for each major opcode. Many fixup requests use a range of opcodes; only a constant for the beginning of the range is defined. The meaning of each fixup request is described below. The opcode ranges and parameters for each fixup are described in the table further below.

`R_NO_RELOCATION`	Copy L bytes with no relocation.
`R_ZEROES`	Insert L zero bytes into the output subspace.
`R_UNINIT`	Skip L bytes in the output subspace.

R_RELOCATION	Copy one data word with relocation. The word is assumed to contain a 32-bit pointer relative to its own subspace.
R_DATA_ONE_SYMBOL	Copy one data word with relocation relative to an external symbol whose symbol index is S.
R_DATA_PLABEL	Copy one data word as a 32-bit procedure label, referring to the symbol S. The original contents of the word should be 0 (no static link) or 2 (static link required).
R_SPACE_REF	Copy one data word as a space reference. This fixup request is not currently supported.
R_REPEATED_INIT	Copy L bytes from the input subspace, replicating the data to fill M bytes in the output subspace.
R_PCREL_CALL	Copy one instruction word with relocation. The word is assumed to be a pc-relative procedure call instruction (for example, BL). The target procedure is identified by symbol S, and the parameter relocation bits are R.
R_ABS_CALL	Copy one instruction word with relocation. The word is assumed to be an absolute procedure call instruction (for example, BLE). The target procedure is identified by symbol S, and the parameter relocation bits are R.
R_DP_RELATIVE	Copy one instruction word with relocation. The word is assumed to be a dp-relative load or store instruction (for example, ADDIL, LDW, STW). The target symbol is identified by symbol S. The linker forms the difference between the value of the symbol S and the value of the symbol $global$. By convention, the value of $global$ is always contained in register 27. Instructions other than LDIL and ADDIL may have a small constant in the displacement field of the instruction.
R_DLT_REL	Copy one instruction word with relocation. The word is assumed to be a register-r18-relative load or store instruction (for example, LDW, LDO, STW). The target symbol is identified by symbol S. The linker computes a linkage table offset relative to register 18 (reserved for a linkage table pointer in position-independent code) for the symbol S.
R_CODE_ONE_SYMBOL	Copy one instruction word with relocation. The word is assumed to be an instruction referring to symbol S (for example, LDIL, LDW, BE). Instructions other than LDIL and ADDIL may have a small constant in the displacement field of the instruction.
R_MILLI_REL	Copy one instruction word with relocation. The word is assumed to be a short millicode call instruction (for example, BLE). The linker forms the difference between the value of the target symbol S and the value of symbol 1 in the module's symbol table. By convention, the value of symbol 1 should have been previously loaded into the base register used in the BLE instruction. The instruction may have a small constant in the displacement field of the instruction.
R_CODE_PLABEL	Copy one instruction word with relocation. The word is assumed to be part of a code sequence forming a procedure label (for example, LDIL, LDO), referring to symbol S. The LDO instruction should contain the value 0 (no static link) or 2 (static link required) in its displacement field.
R_BREAKPOINT	Copy one instruction word conditionally. On HP-UX, the linker always replaces the word with a NOP instruction.
R_ENTRY	Define a procedure entry point. The stack unwind bits, U, and the frame size, F, are recorded in a stack unwind descriptor.
R_ALT_ENTRY	Define an alternate procedure entry point.
R_EXIT	Define a procedure exit point.

R_BEGIN_TRY Define the beginning of a try/recover region.

R_END_TRY Define the end of a try/recover region. The offset R defines the distance in bytes from the end of the region to the beginning of the recover block.

R_BEGIN_BRTAB Define the beginning of a branch table.

R_END_BRTAB Define the end of a branch table.

R_AUX_UNWIND Define an auxiliary unwind table. **CN** is a symbol index of the symbol that labels the beginning of the compilation unit string table. **SN** is the offset, relative to the **CN** symbol, of the scope name string. **SK** is an integer specifying the scope kind.

R_STATEMENT Define the beginning of statement number N.

R_SEC_STATEMENT Define the beginning of a secondary statement number N.

R_DATA_EXPR Pop one word from the expression stack and copy one data word from the input subspace to the output subspace, adding the popped value to it.

R_CODE_EXPR Pop one word from the expression stack, and copy one instruction word from the input subspace to the output subspace, adding the popped value to the displacement field of the instruction.

R_FSEL Use an F´ field selector for the next fixup request instead of the default appropriate for the instruction.

R_LSEL Use an L-class field selector for the next fixup request instead of the default appropriate for the instruction. Depending on the current rounding mode, L´, LS´, LD´, or LR´ may be used.

R_RSEL Use an R-class field selector for the next fixup request instead of the default appropriate for the instruction. Depending on the current rounding mode, R´, RS´, RD´, or RR´ may be used.

R_N_MODE Select round-down mode (L´/R´). This is the default mode at the beginning of each subspace. This setting remains in effect until explicitly changed or until the end of the subspace.

R_S_MODE Select round-to-nearest-page mode (LS´/RS´). This setting remains in effect until explicitly changed or until the end of the subspace.

R_D_MODE Select round-up mode (LD´/RD´). This setting remains in effect until explicitly changed or until the end of the subspace.

R_R_MODE Select round-down-with-adjusted-constant mode (LR´/RR´). This setting remains in effect until explicitly changed or until the end of the subspace.

R_DATA_OVERRIDE Use the constant V for the next fixup request in place of the constant from the data word or instruction in the input subspace.

R_TRANSLATED Toggle "translated" mode. This fixup request is generated only by the linker during a relocatable link to indicate a subspace that was originally read from an old-format relocatable object file.

R_COMP1 Stack operations. The second byte of this fixup request contains a secondary opcode. In the descriptions below, A refers to the top of the stack and B refers to the next item on the stack. All items on the stack are considered signed 32-bit integers.

R_PUSH_PCON1	Push the (positive) constant V.
R_PUSH_DOT	Push the current virtual address.
R_MAX	Pop A and B, then push max(A, B).
R_MIN	Pop A and B, then push min(A, B).
R_ADD	Pop A and B, then push A + B.
R_SUB	Pop A and B, then push B - A.
R_MULT	Pop A and B, then push A * B.

	R_DIV	Pop A and B, then push B / A.
	R_MOD	Pop A and B, then push B % A.
	R_AND	Pop A and B, then push A & B.
	R_OR	Pop A and B, then push A | B.
	R_XOR	Pop A and B, then push A XOR B.
	R_NOT	Replace A with its complement.
	R_LSHIFT	If C = 0, pop A and B, then push B << A. Otherwise, replace A with A << C.
	R_ARITH_RSHIFT	If C = 0, pop A and B, then push B >> A. Otherwise, replace A with A >> C. The shifting is done with sign extension.
	R_LOGIC_RSHIFT	If C = 0, pop A and B, then push B >> A. Otherwise, replace A with A >> C. The shifting is done with zero fill.
	R_PUSH_NCON1	Push the (negative) constant V.
R_COMP2	More stack operations.	
	R_PUSH_PCON2	Push the (positive) constant V.
	R_PUSH_SYM	Push the value of the symbol S.
	R_PUSH_PLABEL	Push the value of a procedure label for symbol S. The static link bit is L.
	R_PUSH_NCON2	Push the (negative) constant V.
R_COMP3	More stack operations.	
	R_PUSH_PROC	Push the value of the procedure entry point S. The parameter relocation bits are R.
	R_PUSH_CONST	Push the constant V.
R_PREV_FIXUP	The linker keeps a queue of the last four unique multi-byte fixup requests; this is an abbreviation for a fixup request identical to one on the queue. The queue index X references one of the four; X = 0 refers to the most recent. As a side effect of this fixup request, the referenced fixup is moved to the front of the queue.	
R_RESERVED	Fixups in this range are reserved for internal use by the compilers and linker.	

The following table shows the mnemonic fixup request type and length and parameter information for each range of opcodes. In the parameters column, the symbol D refers to the difference between the opcode and the beginning of the range described by that table entry; the symbols B1, B2, B3, and B4 refer to the value of the next one, two, three, or four bytes of the fixup request, respectively.

mnemonic	opcodes	length	parameters
R_NO_RELOCATION	0-23	1	L = (D+1) * 4
	24-27	2	L = (D<<8 + B1 + 1) * 4
	28-30	3	L = (D<<16 + B2 + 1) * 4
	31	4	L = B3 + 1
R_ZEROES	32	2	L = (B1 + 1) * 4
	33	4	L = B3 + 1
R_UNINIT	34	2	L = (B1 + 1) * 4
	35	4	L = B3 + 1
R_RELOCATION	36	1	none
R_DATA_ONE_SYMBOL	37	2	S = B1
	38	4	S = B3
R_DATA_PLABEL	39	2	S = B1
	40	4	S = B3
R_SPACE_REF	41	1	none
R_REPEATED_INIT	42	2	L = 4; M = (B1 + 1) * 4
	43	3	L = B1 * 4; M = (B1 + 1) * L
	44	5	L = B1 * 4; M = (B3 + 1) * 4
	45	8	L = B3 + 1; M = B4 + 1
R_PCREL_CALL	48-57	2	R = rbits1(D); S = B1
	58-59	3	R = rbits2(D<<8 + B1); S = B1

	60-61	5	R = rbits2(D<<8 + B1); S = B3
R_ABS_CALL	64-73	2	R = rbits1(D); S = B1
	74-75	3	R = rbits2(D<<8 + B1); S = B1
	76-77	5	R = rbits2(D<<8 + B1); S = B3
R_DP_RELATIVE	80-111	1	S = D
	112	2	S = B1
	113	4	S = B3
R_DLT_REL	120	2	S = B1
	121	4	S = B3
R_CODE_ONE_SYMBOL	128-159	1	S = D
	160	2	S = B1
	161	2	S = B3
R_MILLI_REL	174	2	S = B1
	175	4	S = B3
R_CODE_PLABEL	176	2	S = B1
	177	4	S = B3
R_BREAKPOINT	178	1	none
R_ENTRY	179	9	U,F = B8 (U is 37 bits; F is 27 bits)
	180	6	U = B5 >> 3; F = pop A
R_ALT_ENTRY	181	1	none
R_EXIT	182	1	none
R_BEGIN_TRY	183	1	none
R_END_TRY	184	1	R = 0
	185	2	R = B1 * 4
	186	4	R = B3 * 4
R_BEGIN_BRTAB	187	1	none
R_END_BRTAB	188	1	none
R_STATEMENT	189	2	N = B1
	190	3	N = B2
	191	4	N = B3
R_DATA_EXPR	192	1	none
R_CODE_EXPR	193	1	none
R_FSEL	194	1	none
R_LSEL	195	1	none
R_RSEL	196	1	none
R_N_MODE	197	1	none
R_S_MODE	198	1	none
R_D_MODE	199	1	none
R_R_MODE	200	1	none
R_DATA_OVERRIDE	201	1	V = 0
	202	2	V = B1
	203	3	V = B2
	204	4	V = B3
	205	5	V = B4
R_TRANSLATED	206	1	none
R_AUX_UNWIND	207	12	CU,SN,SK = B11 (CU is 24 bits; SN is 32 bits; SK is 32 bits)
R_COMP1	208	2	OP = B1; V = OP & 0x3f; C = OP & 0x1f
R_COMP2	209	5	OP = B1; S = B3; L = OP & 1; V = ((OP & 0x7f) << 24) | S
R_COMP3	210	6	OP = B1; V = B4; R = ((OP & 1) << 8) | (V >> 16); S = V & 0xffffff
R_PREV_FIXUP	211-214	1	X = D
R_SEC_STMT	215	1	none
R_RESERVED	224-255	-	reserved

Parameter relocation bits are encoded in the fixup requests in two ways, noted as *rbits1* and *rbits2* in the above table. The first encoding recognizes that the most common procedure calls have only general register arguments with no holes in the parameter list. The encoding for such calls is simply the number of

parameters in general registers (0 to 4), plus 5 if there is a return value in a general register.

The second encoding is more complex; the 10 argument relocation bits are compressed into 9 bits by eliminating some impossible combinations. The encoding is the combination of three contributions. The first contribution is the pair of bits for the return value, which are not modified. The second contribution is 9 if the first two parameter words together form a double-precision parameter; otherwise, it is 3 times the pair of bits for the first word plus the pair of bits for the second word. Similarly, the third contribution is formed based on the third and fourth parameter words. The second contribution is multiplied by 40, the third is multiplied by 4, then the three are added together.

Compiler Records

Compiler records are placed in relocatable files by each compiler or assembler to identify the version of the compiler that was used to produce the file. These records are copied into the executable file by the linker, but are strippable. The structure of a compiler record is shown below. All strings are contained in the symbol string table.

```
struct compilation_unit {
    union name_pt      name;
    union name_pt      language_name;
    union name_pt      product_id;
    union name_pt      version_id;
    int                reserved;
    struct sys_clock   compile_time;
    struct sys_clock   source_time;
};
```

SEE ALSO

as_300(1), as_800(1), cc(1), ld(1), nm_300(1), nm_800(1), strip(1), crt0(3), end(3C), a.out_300(4), magic(4).

NAME

acct - per-process accounting file format

SYNOPSIS

```
#include <sys/acct.h>
```

DESCRIPTION

Files produced as a result of calling `acct()` (see *acct*(2)) have records in the form defined by `<sys/acct.h>`, whose contents are:

```
typedef ushort comp_t; /* "floating point":
                          13-bit fraction, 3-bit exponent */
struct  acct {
    char    ac_flag;        /* Accounting flag */
    char    ac_stat;        /* Exit status */
    ushort  ac_uid;         /* Accounting user ID */
    ushort  ac_gid;         /* Accounting group ID */
    dev_t   ac_tty;         /* control typewriter */
    time_t  ac_btime;       /* Beginning time */
    comp_t  ac_utime;       /* acctng user time in clock ticks */
    comp_t  ac_stime;       /* acctng system time in clock ticks */
    comp_t  ac_etime;       /* acctng elapsed time in clock ticks */
    comp_t  ac_mem;         /* memory usage in clicks */
    comp_t  ac_io;          /* chars trnsfrd by read/write */
    comp_t  ac_rw;          /* number of block reads/writes */
    char    ac_comm[8];     /* command name */
};
#define AFORK   01          /* has executed fork, but no exec */
#define ASU     02          /* used super-user privileges */
#define ACCTF   0300        /* record type: 00 = acct */
```

In `ac_flag`, the **AFORK** flag is turned on by each `fork()` and turned off by an `exec()` (see *fork*(2) and *exec*(2)). The `ac_comm` field is inherited from the parent process and is reset by any `exec()`. Each time the system charges the process with a clock tick, it also adds to `ac_mem` the current process size, computed as follows:

(data size) + (text size) + (number of in-core processes sharing text) +
sum of ((shared memory segment size) / (number of in-core processes attached to segment))

For systems with virtual memory, the text, data, and shared memory sizes refer to the resident portion of the memory segments. The value of `ac_mem/(ac_stime+ac_utime)` can be viewed as an approximation to the mean process size, as modified by text-sharing.

The `tacct` structure, which resides with the source files of the accounting commands, represents the total accounting format used by the various accounting commands:

```
/*
* total accounting (for acct period), also for day
*/
struct tacct {
    uid_t           ta_uid;     /* userid */
    char            ta_name[8]; /* login name */
    float           ta_cpu[2];  /* cum. cpu time, p/np (mins) */
    float           ta_kcore[2]; /* cum kcore-minutes, p/np */
    float           ta_con[2];  /* cum. connect time, p/np, mins */
    float           ta_du;      /* cum. disk usage */
    long            ta_pc;      /* count of processes */
    unsigned short  ta_sc;      /* count of login sessions */
    unsigned short  ta_dc;      /* count of disk samples */
    short           ta_fee;     /* fee for special services */
};
```

SEE ALSO

acct(2), acct(1M), acctcom(1M), exec(2), fork(2).

WARNINGS

The `ac_mem` value for a short-lived command gives little information about the actual size of the command because `ac_mem` can be incremented while a different command (such as the shell) is being executed by the process.

STANDARDS CONFORMANCE

`acct`: SVID2, XPG2

NAME

ar - common archive file format

SYNOPSIS

#include <ar.h>

DESCRIPTION

The **ar** command is used to concatenate several files into an archival file (see *ar*(1)). Archives are used mainly as libraries to be searched by the link editor (see *ld*(1).

Each archive begins with the archive magic string.

```
#define  ARMAG   "!<arch>\n"        /* magic string */
#define  SARMAG  8                  /* length of magic string */
```

Each archive which contains object files (see *a.out*(4)) includes an archive symbol table. This symbol table is used by the link editor (see *ld*(1)) to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and/or updated by **ar**.

Following the archive magic string are the archive file members. Each file member is preceded by a file member header which is of the following format:

```
#define AR_NAME_LEN      16

struct ar_hdr   /* archive file member header - printable ascii */
{
    char    ar_name[16];    /* file member name - '/' terminated */
    char    ar_date[12];    /* file member date - decimal */
    char    ar_uid[6];      /* file member user id - decimal */
    char    ar_gid[6];      /* file member group id - decimal */
    char    ar_mode[8];     /* file member mode - octal */
    char    ar_size[10];    /* file member size - decimal */
    char    ar_fmag[2];     /* ARFMAG - string to end header */
};
```

All information in the file member headers is in printable ASCII. The numeric information contained in the headers is stored as decimal numbers (except for **ar_mode** which is in octal). Thus, if the archive contains printable files, the archive itself is printable.

The **ar_name** field is blank-padded and slash (/) terminated. The **ar_date** field is the modification date of the file at the time of its insertion into the archive. Common format archives can be moved from system to system as long as the portable archive command **ar** is used. Note that older versions or **ar** did not use the common archive format, and those archives cannot be read or written by the common archiver.

Each archive file member begins on an even byte boundary; a new-line character is inserted between files if necessary. Nevertheless, the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file. If the archive symbol table exists, the first file in the archive has a zero-length name (i.e., **ar_name[0] == '/'**). The contents of this archive member are machine-dependent. Refer to the appropriate *a.out**(4) manual entry for more information.

SEE ALSO

ar(1), ld(1), strip(1), a.out(4), magic(4).

CAVEATS

strip removes all archive symbol entries from the header (see *strip*(1)). Archive symbol entries must be restored by using the **ts** option of the **ar** command before the archive can be used with the **ld** link editor.

NAME

audeventstab - define and describe audit system events

DESCRIPTION

The `/etc/audeventstab` file lists audit event numbers, corresponding mnemonic names, and brief explanations of each event. Blank lines and comments (beginning with a # character) are allowed. Each non-comment, non-blank line in this file contains three parts:

event	Audit event number in decimal: a single field separated by whitespace.
name	Corresponding mnemonic name: a single field separated by whitespace.
explanation	Remainder of the line, following a # character.

For kernel-generated audit events, event numbers match kernel-internal system call numbers, and event names are system call names. For events from self-auditing programs, names are macros defined in `<sys/audit.h>`.

EXAMPLES

To extract a list of event numbers and names from the file by stripping comments and ignoring blank lines:

```
tab=' '
sed < /etc/audeventstab -e 's/#.*//' -e "/^[ $tab]*$/d"
```

AUTHOR

`audeventstab` was developed by HP.

FILES

`/etc/audeventstab`

SEE ALSO

audisp(1M), audevent(1M).

NAME

audit - file format and other information for auditing

SYNOPSIS

```
#include <sys/audit.h>
```

DESCRIPTION

Audit records are generated when users make security-relevant system calls, as well as by self-auditing processes that call **audwrite()** (see *audwrite*(2)). Access to the auditing system is restricted to super-user.

Each audit record consists of an audit record header and a record body. The record header is comprised of time, process ID, error, event type, and record body length. The time refers to the time the audited event completes in either success or failure; the process ID belongs to the process being audited; the event type is a field identifying the type of audited activity; the length is the record body length expressed in bytes. The exact format of the header is defined in **<sys/audit.h>** as follows:

```
struct audit_hdr {
    u_long ah_time;      /* date/time (tv_sec of timeeval) */
    u_short ah_pid;      /* process ID */
    u_short ah_error;    /* success/failure */
    u_short ah_event;    /* event being audited */
    u_short ah_len;      /* length of variant part */
};
```

The record body is the variable-length component of an audit record containing more information about the audited activity. For records generated by system calls, the body contains the parameters of the system calls; for records generated by self-auditing processes, the body consists of a high-level description of the event (see *audwrite*(2)).

The records in the audit file are compressed to save file space. When a process is audited the first time, a *pid* identification record (PIR) is written into the audit file containing information that remains constant throughout the lifetime of the process. This includes the parent's process ID, audit ID, real user ID, real group ID, effective user ID, effective group ID, and the terminal ID (tty). The PIR is entered only once per process per audit file, and is also defined in **<sys/audit.h>** as follows:

```
struct pir_body {          /* pir-related info */
    short   ppid;          /* parent process ID */
    aid_t   aid;           /* audit ID */
    u_short ruid;          /* user_ID */
    u_short rgid;          /* group ID */
    u_short euid;          /* effective user_ID */
    u_short egid;          /* effective group_ID */
    dev_t   tty;           /* tty number */
};
```

Information accumulated in an audit file is analyzed and displayed by **audisp** (see *audisp*(1M)).

Whenever auditing is turned on, a "current" audit file is required and a "next" audit file (for backup) is recommended (see *audsys*(1M) and *audomon*(1M)). When the "current" audit file is full and the "next" audit file is available, the auditing system switches files automatically.

AUTHOR

audit was developed by HP.

SEE ALSO

audsys(1M), audevent(1M), audisp(1M), audomon(1M), audwrite(2), getevent(2), setevent(2).

NAME

bif - Bell Interchange Format utilities

DESCRIPTION

BIF (Bell Interchange Format) is the name given to the format of mounted media used by HP 9000 Series 200 HP-UX releases 2.0 and 2.1, and by the HP Integral Personal Computer. This format is based upon that used in AT&T System III UNIX.

The BIF utilities listed under SEE ALSO below are provided for non-Series 200 (revision 2.0 and 2.1) and non-Integral Personal Computer access to the BIF media. These utilities read and write data to and from BIF volumes, as well as retrieve and store information on BIF volumes.

The BIF utilities listed below are the only HP-UX utilities that recognize the internal contents of a BIF volume. To the rest of HP-UX, a BIF volume is simply a file or disk containing unspecified data. Therefore, *do not use* **mount** *on a BIF volume*; the operating system cannot recognize it.

BIF file names are specified to the BIF utilities by concatenating the HP-UX path name for the BIF volume with the BIF file name, separating the two with a colon (**:**). For example,

/dev/rdsk/1s0:/users/ivy
specifies BIF file **/users/ivy** on HP-UX device special file **/dev/rdsk/1s0**

Note that this file naming convention is applicable only for use as arguments to the BIF utilities and does not constitute a legal path name for any other use within the HP-UX operating environment. Shell (see *sh*(1)) "meta" characters *****, **?**, and **[...]** do not work for specifying an arbitrary pattern for file-name matching when using BIF utilities.

If the device name and a trailing colon are specified without a file or directory name following, as in **/dev/rdsk/1s0:**, the root (**/**) of the BIF file system is assumed by convention.

A primitive form of data protection is provided by a lockfile **/tmp/BIF.LCK** that only allows one process and its immediate children to use the BIF utilities at any given time.

AUTHOR

bif utilities were developed by HP.

SEE ALSO

bifchmod(1), bifchown(1), bifcp(1), bifdf(1M), biffind(1), biffsck(1M), biffsdb(1M), bifls(1), bifmkdir(1), bifmkfs(1M), bifrm(1).

NAME

cdf - context dependent files

DESCRIPTION

A context dependent file (CDF) consists of several files grouped under the same path name. The system ordinarily selects one of the files using the context of the process (see *context*(5)). This mechanism allows machine dependent executable, system data and device files to work correctly from all nodes in a cluster while using the same path name.

A CDF is implemented as a special kind of directory, marked by a bit in its mode (see *chmod*(2)). The name of the CDF is the name of the directory; the contents of the directory are files with names that are expected to match one part of a process context. When such a directory is encountered during a path name search, the names of the files in the directory are compared with each string in the process's context, in the order in which the strings appear in the context. The *first* match is taken to be the desired file. The name of the directory thus refers to one of the files within it, and the directory itself is normally invisible to the user. Hence, the directory is called a **hidden directory**.

When a process with a context that does not match any file names in the hidden directory attempts to access a file by the path name of the hidden directory, the reference is unresolved; no file with that path name appears to exist. When such a process attempts to create a file with the path name of the hidden directory, it creates within the hidden directory a file whose name is the cnode name from the process's context.

A hidden directory itself can be accessed explicitly, overriding the normal selection according to context, by appending the character + to the directory's file name.

EXAMPLES

Consider a cluster with three versions of **/etc/inittab**: one for cnode **william**, one for cnode **david**, and a common file for the rest of the cnodes. The contents of the hidden directory **/etc/inittab**, as shown by the command:

```
ls -l /etc/inittab+
```

would then be:

```
-rwxr-xr-x'1'root'other'1416'Mar  7 10:08'david
-rwxr-xr-x'1'root'other'1211'Apr 12 11:16'default
-rwxr-xr-x'1'root'other'1037'Apr  3 12:04'william
```

The file names **william** and **david** match the cnode name in the context of all processes on those cnodes. The file named **default** matches all contexts if no other file in the hidden directory matches, and thus matches the contexts of all processes on other cnodes. While a **default** file may appear in a hidden directory, it is not necessary to have one. If it did not exist in this case, nodes other than **william** and **david** would not see any file named **/etc/inittab**.

If a user on the cnode **william** wants to find the difference between the local cnode's **/etc/inittab** and the default version, any of the commands:

```
diff /etc/inittab /etc/inittab+/default
diff /etc/inittab+/william /etc/inittab+/default
```

or

```
cd /etc/inittab+ ; diff william default
```

refer to the appropriate files.

A directory is changed to a hidden directory by using the **chmod** command (see *chmod*(1)). In the above example, if **/etc/inittab** was an ordinary directory,

```
chmod +H /etc/inittab
```

would be used to make it a hidden directory. Invoking:

```
chmod -H /etc/inittab+
```

makes **/etc/inittab** appear as a regular directory. Note that the + must appear so that the mode will be changed for the hidden directory itself.

A hidden directory can contain files of any type, including other CDFs. Such use of nested CDFs is preferable to relying on the order of items in the context for selecting files from a single hidden directory. For example, in a cluster of HP 9000 Series 300 workstations, some of the workstations might have HP 98248A floating-point accelerators, and one of those, with a cnode name of `color`, might have a different graphics display. If `/usr/local/bin/graphicsprog` is a floating-point-intensive application that uses the local graphics display, it might be useful to have three versions built, each for the appropriate hardware configurations. The arrangement of the three versions, as shown by the command:

```
ls -lRH /usr/local/bin/graphicsprog+
```

might then be:

```
total 202 dr-sr-xr-x'2'bin'bin'1024'Feb 26 17:34'HP98248A+ -r-xr-xr-
x'1'bin'bin'101144'Feb 26 17:31'default

graphicsprog+/HP98248A+: total 414 -r-xr-xr-x'1'bin'bin'105112'Feb 26
17:34'color -r-xr-xr-x'1'bin'bin'103732'Feb 26 17:40'default
```

AUTHOR

`cdf` was developed by HP.

SEE ALSO

chmod(1), find(1), getcontext(1), ls(1), pwd(1), showcdf(1), tar(1). makecdf(1M), chmod(2), getcontext(2), getcdf(3C), context(5),

NAME

CDFinfo - CDFinfo file format and rule syntax

DESCRIPTION

Each fileset that is loaded and maintained on a HP-UX cluster server may contain a **CDFinfo** file located in **/system/***fileset_name***/CDFinfo**. The **CDFinfo** file contains instructions on how context-dependent files and directories in that fileset should be handled by **update** and **sam** (see *update*(1M) and *sam*(1M)).

See the WARNINGS section below for issues concerning future support of the **CDFinfo** mechanism of cluster maintenance.

Filesets containing files that cannot be shared by every system in an HP-UX cluster need some way of specifying how these files should be handled. The **CDFinfo** file contains rules specifying which files should be Context-Dependent Files (see *cdf*(4)).

It is possible to construct a product that does not require the use of CDFs to operate properly in a clustered environment. The use of front-end shell scripts that invoke the correct executable is one example of how to eliminate the need for CDFs. A product that does not require CDFs to operate are easier to maintain in a multi-vendor environment, and are less likely to need modification if the cluster support mechanisms change (see WARNINGS).

The **CDFinfo** file is used during three different operations on a clustered system. They are:

- When creating a cluster for the first time (using **sam**).
- When adding a client (using **sam**).
- When loading a fileset onto a clustered system (using **update**).

For each file or directory in a fileset that cannot be shared by all systems in a cluster (executable files or per-cnode configurable files, for example), a rule can be specified in the **CDFinfo** file that indicates how that file should be handled in each of the three cases above. Files that can be shared by all cnodes do not need an entry in the **CDFinfo** file (text documents and shell scripts are good examples). Also, files whose path contains a directory that already has a **CDFinfo** entry do not need an individual entry, as long as the directory is the correct type of CDF. Thus, executable files under **/bin** or **/usr/bin** do not need an individual architecture-specific CDF. Likewise, client-specific files under the **/usr/adm** directory do not need their own cnode-specific **CDFinfo** entry.

The first line of the **CDFinfo** file must contain the string:

*fileset_name*

Where *fileset_name* is the name of the fileset to which the **CDFinfo** file belongs.

The rest of the file contains rules that conform to the following general syntax:

```
path_name {
        createcdf  (
                action  [[; action ]...]
        )
        addcnode  (
                action  [[; action ]...]
        )
        update  (
                action  [[; action ]...]
        )
}
```

path_name must beginning with **/**. Any or all of the **createcdf**, **addcnode**, or **update** portions of a rule can be specified. A file that is an architecture CDF does not need to contain the **addcnode** portion.

The *action* contained in the **createcdf** portion of a rule is performed when **sam** is used to create a cluster server for the first time. It can also be used by **update** if this rule is referenced by a **create** action (see the **create** action for details).

The *action* contained in the **addcnode** portion of a rule is performed when **sam** is used to add a client to an existing cluster system. It can also be used by **update** if this rule is referenced by a **create** action (see the **create** action for details).

The *action* contained in the **update** portion of a rule is performed when **update** is used to load this fileset onto an existing cluster system. Only the CDF rules for files that are loaded from the media are exercised. Therefore there must be a file (or directory) on the media with the exact name: *path_name* for that rule to be triggered. This differs from the **createcdf** and **addcnode** portions of rules, which are all executed when **sam** creates a cluster or adds a client-node.

Actions

Each *action* has the syntax:

operator [*argument* [*argument*]]

Valid *operators* are described in the next section. *arguments* can be constructed by combining one or more of the following elements (see specific *actions* for restrictions):

"quoted-string"	A double-quoted string (usually a file path). No substitutions are made between the quotes.
ARCH	When **ARCH** is specified in an *argument*, the string **ARCH** is replaced by the CDF context element for the hardware architecture on which the file is to be used. The architecture of the file is determined from the file's magic number (if possible), or defaults to the appropriate architecture of the the system (in the case of **createcdf** or **addcnode** rule portions), or to the architecture of the update media being loaded (in the case of an **update** rule portion). Possible values include **HP-PA** (Series 700 and 800) or **HP-MC68020** (Series 300 and 400). Actions that contain **ARCH** are ignored on a homogeneous cluster. Currently Series 300/400 servers are defined as homogeneous (can contain only Series 300/400 cnodes). Where as Series 700/800 servers are defined as heterogeneous (can contain Series 300/400 clients). Architecture-specific CDFs are only created on heterogeneous cluster servers.
CNODE	The string **CNODE** is replaced by the name of the client-node being added. **CNODE** can only be used in the **addcnode** portion of a rule.
LOCALROOT	The string **LOCALROOT** is replaced by the **localroot** context element. The **localroot** element of a CDF is accessed by the cluster server. A CDF containing **localroot** is usually complemented with the addition of a **remoteroot** element.
REMOTEROOT	The string **REMOTEROOT** is replaced by the **remoteroot** context element. The **remoteroot** element of a CDF is accessed by all non-server cnodes. A CDF containing **remoteroot** is usually complemented with the addition of a **localroot** element.
ROOTSERVER	The string **ROOTSERVER** is replaced by the cnode name of the cluster server.
ROOTARCH	The string **ROOTARCH** is replaced by the architecture string (**HP-PA** or **HP-MC68020**) that corresponds to the cluster server.

Operators

The following *operators* can be used to make up the rule's *action*:

save Usage: **save** *destination* [*alternate*]
Example: **save ARCH "/etc/newconfig/foo"**

The **save** operator may be used only in the **createcdf** portion of a rule, and converts an existing file or directory (*path_name*) into a CDF. In general, the existing file (or directory) becomes an element of the CDF (as specified by the *destination*). If *path_name* does not exist, the contents of the file identified by *alternate* are copied into the element of the newly created CDF. The *destination* argument identifies the name of the element of the CDF to be created. It can only be one of: **ARCH**, **LOCALROOT**, **REMOTEROOT**, **ROOTSERVER**, or **ROOTARCH**. The **save** action translates directly into the HP-UX command:

makecdf -c *destination* [**-f** *alternate*] *path_name*

load Usage: **load** *destination*

The **load** operator can be used only within the **update** portion of a rule and is only exercised when **update** extracts a file (or directory) called *path_name*. If *path_name* is a regular file, a CDF is created and the file is loaded as the CDF element specified by *destination*. If *path_name* is a directory, a CDF is created, and a directory is created as the CDF element as specified by *destination*. When a directory is created as an architecture-specific CDF both **HP-PA** and **HP-MC68020** context elements are created, even if one element will remain empty.

exec Usage: **exec "command_path"** [*arg* ...]
Example: **exec "/bin/echo" CNODE ">/etc/issue+"/CNODE**

The **exec** operator can be used only in the **createcdf** and **addcnode** portions of a rule. **exec** can be used where the required operation is too complex to be handled by the existing CDF rules, making it necessary to use an external command to accomplish the desired action. Any number of arguments can be provided. Use double-quotes for strings that must not be altered. Any of the *argument* specifiers listed previously (**CNODE**, **ARCH**, etc.) can be used as arguments. These arguments are then passed to *command_path* when invoked.

copy Usage: **copy** [*source*] *destination*
Example 1: **copy "/etc/newconfig/inittab" CNODE**
Example 2: **copy ROOTSERVER CNODE**

The **copy** action is used primarily in the **addcnode** portion of rules to copy a file (or directory) into the CDF element of a newly configured client, but can be used in all rule portions if needed. In example 1 above, it is used to copy the configurable file: **/etc/newconfig/inittab** into the **CNODE** element of *path_name*. In example 2, it is used to copy the file located from the **ROOTSERVER** CDF element into the **CNODE** element.

When used in the **update** portion of a rule, the *source* is implied to be that of *path_name* and can be omitted.

When the **copy** action is applied to a directory, only the subdirectory structure is copied; no regular files are copied. The intent of this is to support operations such as creating **/usr/spool/cron** and its subdirectories for a new cnode using the cluster server's **/usr/spool/cron** directory as a template for what directories should exist.

symlink The **symlink** operator is identical to the **copy** operator, except that the *destination* file is symbolically linked to the *source* rather than copied.

move The **move** operator is identical to the **copy** operator, except that the *source* file is moved to the *destination* rather than copied.

remove Usage: **remove**

The **remove** operator may be used only in the **createcdf** and **update** portions of a rule. In the **createcdf** portion, it is used to specify that the CDF should be created, but the standalone file should be removed instead of being copied into a CDF element. This is useful for run-time data a program can create (such as **/etc/ps_data**). In the **update** portion, it is used to specify that the file should not be loaded from the media.

create Usage: **create** *path_name*

The **create** operator may be used only in the **update** portion of a rule. **create** is used to trigger the **createcdf** and **addcnode** portions of the rule associated with *path_name*. This is the only way to exercise the **createcdf** and **addcnode** rule portions outside of the initial cluster creation and cnode addition done by **sam**.

See WARNINGS below for details on a problem concerning the **create** operator.

Because of its complex nature, the **create** operator should be avoided if possible.

NAME

cdfs - format of CDFS file system volume

SYNOPSIS

```
#include <sys/types.h>
#include <sys/cdfs.h>
#include <sys/cdfsdir.h>
```

DESCRIPTION

Each CD-ROM can contain one or more volumes. Each of these volumes can contain a CDFS file system (see *cdrom*(4) for a description of the overall format of a CD-ROM).

The attributes of a CDFS volume are described by a *volume descriptor*. (Note that only the primary volume descriptor is recognized and used at the current level of CD-ROM support. Thus, "volume descriptor" in the following discussion refers to the primary volume descriptor.) A volume descriptor is 2048 bytes in length and is described by two data structures, one for HSG (High Sierra Group) format, and one for ISO-9660 (International Organization for Standardization) format. Only the pertinent portions of `sys/cdfs.h` are reproduced here:

```
#define KMAXNAMLEN          30+2+5
#define VOL_SET_ID_SIZ      128
#define VOL_ID_SIZ          32
#define STD_ID_SIZ          5
#define SYS_ID_SIZ          32
#define PUBLISHER_ID_SIZ    128
#define PREPARER_ID_SIZ     128
#define APPLICATION_ID_SIZ  128
#define APPL_USE_SIZ        512

struct icdfs            /*primary volume descriptor-ISO-9660*/
{
    char cdf_vold_type;             /*volume descriptor type*/
    char cdf_std_id[STD_ID_SIZ];    /*id "CD001" for ISO-9660*/
    char cdf_vold_version;          /*should be 1 for ISO-9660*/
    char cdf_unused1;               /*spare*/
    char cdf_sys_id[SYS_ID_SIZ];    /*id of a system that knows contents of
                                      system area, logic sector 0-15*/
    char cdf_vol_id[VOL_ID_SIZ];    /*id of this volume*/
    char cdf_unused2[8];            /*spare*/
    int  cdf_vol_size_lsb;          /*size (LSB) of volume in logic block*/
    int  cdf_vol_size_msb;          /*size (MSB) of volume in logic block*/
    char cdf_unused3[32];           /*spare*/
    ushort cdf_volset_siz_lsb;      /*size (LSB) of volume set*/
    ushort cdf_volset_siz_msb;      /*size (MSB) of volume set*/
    ushort cdf_volset_seq_lsb;      /*sequence no. (LSB) of vol in the set*/
    ushort cdf_volset_seq_msb;      /*sequence no. (MSB) of vol in the set*/
    ushort cdf_logblk_siz_lsb;      /*size (LSB) of logic block in bytes*/
    ushort cdf_logblk_siz_msb;      /*size (MSB) of logic block in bytes*/
    u_int  cdf_pathtbl_siz_lsb;     /*size (LSB) of path table in bytes */
    u_int  cdf_pathtbl_siz_msb;     /*size (MSB) of path table in bytes */
    u_int  cdf_pathtbl_loc_lsb;     /*logical block no. (LSB) of path table*/
    u_int  cdf_pathtblo_loc_lsb;    /*logical block no. (LSB) of
                                      optional path table*/
    u_int  cdf_pathtbl_loc_msb;     /*logical block no. (MSB) of path table*/
    u_int  cdf_pathtblo_loc_msb;    /*logical block no. (MSB) of
                                      optional path table*/
    struct min_cddir cdf_rootdp;          /*directory record of root*/
    char cdf_vol_set_id[VOL_SET_ID_SIZ]; /*id of the volume set*/
    char cdf_pb_id[PUBLISHER_ID_SIZ];     /*publisher's id*/
    char cdf_pp_id[PREPARER_ID_SIZ];      /*preparer's id*/
    char cdf_ap_id[APPLICATION_ID_SIZ];   /*application id*/
```

```
    char cdf_copyright[KMAXNAMLEN];      /*copyright in this file under
                                           root directory, the max.
                                           len. is 18, the rest unused*/
    char cdf_abstract[KMAXNAMLEN];       /*abstract in this file under
                                           root directory, the max.
                                           len. is 18, the rest unused*/
    char cdf_bibliographic[KMAXNAMLEN];  /*bibliographic in this file
                                           under root directory, the max.
                                           len. is 18, the rest unused*/
/*The next four chunks are creation time, modification time, expiration time
  and effective time.  Since the date/time info is 17 bytes(odd), a structure
  can't be used (compiler rounds up to even bytes).  If for any reason this
  info is changed, make sure to fix all four of them.*/
/*creation time*/
    char cdf_c_year[4];          /*years since year 0000*/
    char cdf_c_month[2];         /*month*/
    char cdf_c_day[2];           /*day*/
    char cdf_c_hour[2];          /*hour*/
    char cdf_c_minute[2];        /*minute*/
    char cdf_c_second[2];        /*second*/
    char cdf_c_h_second[2];      /*hundredths of second*/
    char cdf_c_timezone;         /*timezone, offset from Greenwich Mean Time
                                   in number of 15 minutes intervals from
                                   -48(West) to +52(East)*/
/*modification time*/
    char cdf_m_year[4];          /*years since year 0000*/
    char cdf_m_month[2];         /*month*/
    char cdf_m_day[2];           /*day*/
    char cdf_m_hour[2];          /*hour*/
    char cdf_m_minute[2];        /*minute*/
    char cdf_m_second[2];        /*second*/
    char cdf_m_h_second[2];      /*hundredths of second*/
    char cdf_m_timezone;         /*timezone, offset from Greenwich Mean Time
                                   in number of 15 minutes intervals from
                                   -48(West) to +52(East)*/
/*expiration time*/
    char cdf_x_year[4];          /*years since year 0000*/
    char cdf_x_month[2];         /*month*/
    char cdf_x_day[2];           /*day*/
    char cdf_x_hour[2];          /*hour*/
    char cdf_x_minute[2];        /*minute*/
    char cdf_x_second[2];        /*second*/
    char cdf_x_h_second[2];      /*hundredths of second*/
    char cdf_x_timezone;         /*timezone, offset from Greenwich Mean Time
                                   in number of 15 minutes intervals from
                                   -48(West) to +52(East)*/
/*effective time*/
    char cdf_e_year[4];          /*years since year 0000*/
    char cdf_e_month[2];         /*month*/
    char cdf_e_day[2];           /*day*/
    char cdf_e_hour[2];          /*hour*/
    char cdf_e_minute[2];        /*minute*/
    char cdf_e_second[2];        /*second*/
    char cdf_e_h_second[2];      /*hundredths of second*/
    char cdf_e_timezone;         /*timezone, offset from Greenwich Mean Time
                                   in number of 15 minutes intervals from
                                   -48(West) to +52(East)*/

    u_char cdfs_fs_version;          /*file structure version:1 for ISO-9660*/
```

EXAMPLES

The following example `CDFinfo` file uses several of the rules and actions discussed. For a much larger set of examples, refer to existing `CDFinfo` files located in `/system/*/CDFinfo`.

```
# MY-EXAMPLE

# Make the directory /bin an architecture-specific CDF
# so that files under /bin do not need individual CDF entries.
/bin {
     createcdf (
          save ARCH
     )
     update (
          load ARCH
     )
}

# Make the file /etc/mount an architecture-specific CDF
/etc/mount {
     createcdf (
          save ARCH
     )
     update (
          load ARCH
     )
}

# Make /etc/inittab a CNODE specific CDF, use the new version
# in /etc/newconfig if it doesn't exist, or when adding a client.
/etc/inittab {
     createcdf (
          save ROOTSERVER "/etc/newconfig/inittab"
     )
     addcnode (
          copy "/etc/newconfig/inittab" CNODE
     )
}

#  Make /etc/issue a CNODE specific CDF, and put the CNODE name
#  into the new client's version when adding a cnode.
/etc/issue {
     createcdf (
          save ROOTSERVER
     )
     addcnode (
          exec "/bin/echo" CNODE ">/etc/issue+"/CNODE
     )
}
```

WARNINGS

The information contained is this manual entry pertains specifically to HP-UX releases 7.0 through 9.0. The CDFs and `CDFinfo` files are proprietary to HP-UX systems and it is likely that future releases will support new, different, and more sophisticated mechanisms for file sharing. Therefore products which use the `CDFinfo` mechanism to support clustered systems may need to be revised. For this reason, HP discourages the use of CDFs and `CDFinfo` files when avoidable.

In earlier versions of `update` (before HP-UX release 9.0), the `create` operator does not exercise the `addcnode` nor `createcdf` rule portions if *path_name* already exists on the filesystem (either as a CDF or regular file). `update`'s failure to exercise these rule portions usually causes only one set of cnode elements to be created (the set of cnodes belonging to the same architecture as the media loaded for the first time). This is a problem only on heterogeneous clusters. The `customize` script executed by `update` is usually responsible for adding the remaining cnode elements.

Some of the less commonly used *operators* are known to give unexpected results, especially when applied to filesets that have been relocated (not loaded relative to **/** by **update**). It is suggested that **CDFinfo** files use only the basic *operators* when possible. Try to avoid the use of the **create**, **symlink**, **remove**, **move**, and **copy** *operators* when possible.

Because of the complex nature of the **CDFinfo** file and the processes that use them, testing is a vital element in creating a product that can be successfully used in a clustered environment. When testing a product, be sure to go through the procedures that make use of the **CDFinfo** file:

- With the product already installed on a standalone system, use **sam** to create a cluster for the first time.
- With the product already installed on a cluster system, use **sam** to add a new client.
- Use **update** to load and reload a fileset onto a clustered system. Be sure that all remnants of the fileset (including empty CDFs) have been removed before testing a fresh load.
- Perform the above testing on both heterogeneous (Series 700/800 systems) and homogeneous (Series 300/400 systems) cluster servers.

FILES

/system/*/CDFinfo **CDFinfo** files for installed filesets.

SEE ALSO

fpkg(1M), makecdf(1M), sam(1M), update(1M), cdf(4).

```
    char cdf_unused4;
    char cdf_appl_use[APPL_USE_SIZ]; /*reserved for application*/
    char cdf_future_use[653];        /*reserved for future. Note that if
                                       total size of field before this one
                                       is changed, this size should be
                                       changed so that the size of this
                                       structure is 2048*/
};

struct hcdfs          /*primary volume descriptor-HSG*/
{
    u_int cdf_loc_lsb;             /*logical block no. (LSB) of this descr.*/
    u_int cdf_loc_msb;             /*logical block no. (MSB) of this descr.*/
    char cdf_vold_type;            /*volume descriptor type*/
    char cdf_std_id[STD_ID_SIZ];   /*id "CDROM" for HSG */
    char cdf_vold_version;         /*should be 1 for HSG */
    char cdf_unused1;              /*spare*/
    char cdf_sys_id[SYS_ID_SIZ];   /*id of a system that knows contents of
                                     system area, logical sector 0-15*/
    char cdf_vol_id[VOL_ID_SIZ];   /*id of this volume*/
    char cdf_unused2[8];           /*spare*/
    int  cdf_vol_size_lsb;         /*size (LSB) of volume in logical block*/
    int  cdf_vol_size_msb;         /*size (MSB) of volume in logical block*/
    char cdf_unused3[32];          /*spare*/
    ushort cdf_volset_siz_lsb;     /*size (LSB) of volume set*/
    ushort cdf_volset_siz_msb;     /*size (MSB) of volume set*/
    ushort cdf_volset_seq_lsb;     /*sequence no. (LSB) of volume in the set*/
    ushort cdf_volset_seq_msb;     /*sequence no. (MSB) of volume in the set*/
    ushort cdf_logblk_siz_lsb;     /*size (LSB) of logical block in bytes*/
    ushort cdf_logblk_siz_msb;     /*size (MSB) of logical block in bytes*/
    u_int  cdf_pathtbl_siz_lsb;    /*size (LSB) of path table in bytes */
    u_int  cdf_pathtbl_siz_msb;    /*size (MSB) of path table in bytes */
    u_int  cdf_pathtbl_loc_lsb;    /*logical block no. (LSB) of path table*/
    u_int  cdf_pathtblo1_loc_lsb;  /*logical block number (LSB) of
                                     optional path table*/
    u_int  cdf_pathtblo2_loc_lsb;  /*logical block number (LSB) of
                                     optional path table*/
    u_int  cdf_pathtblo3_loc_lsb;  /*logical block number (LSB) of
                                     optional path table*/
    u_int  cdf_pathtbl_loc_msb;    /*logic block num. (MSB) of path table*/
    u_int  cdf_pathtblo1_loc_msb;  /*logic block num (MSB) of optional
                                     path table*/
    u_int  cdf_pathtblo2_loc_msb;  /*logic block num (MSB) of optional
                                     path table*/
    u_int  cdf_pathtblo3_loc_msb;  /*logic block num (MSB) of optional
                                     path table*/
    struct min_cddir cdf_rootdp;    /*directory record of root*/
    char cdf_vol_set_id[VOL_SET_ID_SIZ];  /*id of the volume set*/
    char cdf_pb_id[PUBLISHER_ID_SIZ];     /*publisher's id*/
    char cdf_pp_id[PREPARER_ID_SIZ];      /*preparer's id*/
    char cdf_ap_id[APPLICATION_ID_SIZ];   /*application id*/
    char cdf_copyright[KMAXNAMLEN-5];     /*copyright in this file under
                                            root directory, the max.
                                            len. is 12, the rest unused*/
    char cdf_abstract[KMAXNAMLEN-5];      /*abstract in this file under
                                            root directory, the max.
                                            len. is 12, the rest unused*/
/* the next four chunks are creation time, modification time,
   expiration time and effective time.  Since the date/time info
```

```
  is 17 bytes(odd), a structure can't be used (compiler rounds up
  to even bytes).  If for any reason this info is changed, be sure
  to fix all four of them. */
/*creation time*/
    char cdf_c_year[4];         /*years since year 0000*/
    char cdf_c_month[2];        /*month*/
    char cdf_c_day[2];          /*day*/
    char cdf_c_hour[2];         /*hour*/
    char cdf_c_minute[2];       /*minute*/
    char cdf_c_second[2];       /*second*/
    char cdf_c_h_second[2];     /*hundredths of second*/
/*modification time*/
    char cdf_m_year[4];         /*years since year 0000*/
    char cdf_m_month[2];        /*month*/
    char cdf_m_day[2];          /*day*/
    char cdf_m_hour[2];         /*hour*/
    char cdf_m_minute[2];       /*minute*/
    char cdf_m_second[2];       /*second*/
    char cdf_m_h_second[2];     /*hundredths of second*/
/*expiration time*/
    char cdf_x_year[4];         /*years since year 0000*/
    char cdf_x_month[2];        /*month*/
    char cdf_x_day[2];          /*day*/
    char cdf_x_hour[2];         /*hour*/
    char cdf_x_minute[2];       /*minute*/
    char cdf_x_second[2];       /*second*/
    char cdf_x_h_second[2];     /*hundredths of second*/
/*effective time*/
    char cdf_e_year[4];         /*years since year 0000*/
    char cdf_e_month[2];        /*month*/
    char cdf_e_day[2];          /*day*/
    char cdf_e_hour[2];         /*hour*/
    char cdf_e_minute[2];       /*minute*/
    char cdf_e_second[2];       /*second*/
    char cdf_e_h_second[2];     /*hundredths of second*/

    u_char cdfs_fs_version;          /*file structure version:1 for HSG*/
    char cdf_unused4;
    char cdf_appl_use[APPL_USE_SIZ]; /*reserved for application*/
    char cdf_future_use[680];        /*reserved for future. Note that if
                                       total size of field before this one
                                       is changed, this size should be
                                       changed so that the size of this
                                       structure is 2048*/
};
```

Note the physical differences between the two formats' volume descriptors:

1. The HSG volume descriptor includes the logical block number at which the descriptor occurs; ISO does not.
2. The HSG volume descriptor provides for more optional path tables (three optional type-L path tables plus three optional type-M path tables) than does ISO (one optional path table of each type);
3. The HSG volume descriptor does not provide for a bibliographic file, while ISO does;
4. The HSG volume descriptor does not provide for GMT offsets in the creation, modification, expiration, and effective times, while ISO does;
5. The HSG volume descriptor's "reserved for future use" field is larger than that of the ISO descriptor's.

Fields are defined as follows:

cdf_loc_lsb Defined for HSG format only. Logical block number at which this particular volume descriptor is arranged in least-significant-byte-first order.

cdf_loc_msb Same as **cdf_loc_lsb** except that volume descriptor is arranged in most-significant-byte-first order.

cdf_vold_type Volume descriptor type. Value should be one (1) for both formats.

cdf_std_id A five-character standard ID which identifies the standard format in use. For ISO, this string is **CD001**; for HSG, it is **CDROM**. Note that this field is the one most often used to differentiate between the two formats.

cdf_vold_version Volume descriptor version number for the volume. This value should be one (1) for both formats.

cdf_sys_id Identifies what system or systems specify and understand the contents of the system area (logical blocks zero through 15). This field can contain a string of up to 32 characters. The ID is left-justified in the field and padded on the right with spaces.

cdf_vol_id Identifies the volume. This field can contain a string of up to 32 characters. The ID is left-justified in the field and padded on the right with spaces.

cdf_vol_size_lsb Size of this particular volume in logical blocks. Value is given in least-significant-byte-first order.

cdf_vol_size_msb Same as **cdf_vol_size_lsb** except that value is given in most-significant-byte-first order,

cdf_volset_siz_lsb
Volume-set size of the volume set of which this volume is a member. This value is given in least-significant-byte-first order.

cdf_volset_siz_msb
Same as **cdf_volset_siz_lsb** except value is in most-significant-byte-first order.

cdf_volset_seq_lsb
Gives the volume-sequence number of this volume in least-significant-byte-first order. The volume-sequence number determines the order in which volumes occur in the volume set.

cdf_volset_seq_msb
Same as **cdf_volset_seq_lsb** except value is in most-significant-byte-first order.

cdf_logblk_siz_lsb
Logical block size for this volume in least-significant-byte-first order. The logical block size cannot be less than 512 bytes, and cannot exceed the logical sector size of the device (currently does not exceed 2048 bytes).

cdf_logblk_siz_msb
Same as **cdf_logblk_siz_lsb** except value is in most-significant-byte-first order.

cdf_pathtbl_siz_lsb
Size of the path table for this volume in bytes. This value is given in least-significant-byte-first order. A path table defines the directory hierarchy of the file system on the volume. The first entry in the table describes the root directory.

cdf_pathtbl_siz_msb
Same as **cdf_pathtbl_siz_lsb** except that value is in most-significant-byte-first order.

`cdf_pathtbl_loc_lsb`
: Logical block number of the block where the mandatory type-L path table begins. This value is given in least-significant-byte-first order. A type-L path table is a path table whose numerical values are all specified in least-significant-byte-first order.

`cdf_pathtblo_loc_lsb`
: (in ISO format)

`cdf_pathtblo1_loc_lsb`,
`cdf_pathtblo2_loc_lsb`,
`cdf_pathtblo3_loc_lsb`
: (in HSG format)
These fields specify the logical block numbers of the blocks where optional type-L path tables begin. If an optional type-L path table does not exist, the corresponding field is given a zero value. All values are specified in least-significant-byte-first order.

`cdf_pathtbl_loc_msb`
: Logical block number of the block where the mandatory type-M path table begins. This value is given in most-significant-byte-first order. A type-M path table is a path table whose numerical values are all specified in most-significant-byte-first order.

`cdf_pathtblo_loc_msb`
: (in ISO format)

`cdf_pathtblo1_loc_msb`,
`cdf_pathtblo2_loc_msb`,
`cdf_pathtblo3_loc_msb`
: (in HSG format)
These fields specify the logical block numbers of the blocks where optional type-M path tables begin. If an optional type-M path table does not exist, the corresponding field is given a zero value. All values are specified in most-significant-byte-first order.

`cdf_rootdp`
: Structure containing a duplicate of the root directory record. This structure is defined in <`sys/cdfsdir.h`> as follows:

```
struct min_cddir {
    u_char  mincdd_reclen;      /*length of directory record in bytes*/
    u_char  mincdd_xar_len;     /*length of XAR in logic blocks*/
    u_int   mincdd_loc_lsb;     /*logic block number of the extent in LSB*/
    u_int   mincdd_loc_msb;     /*logic block number of the extent in MSB*/
    u_int   mincdd_size_lsb;    /*size (in bytes) of the file section in LSB*/
    u_int   mincdd_size_msb;    /*size (in bytes) of the file section in MSB*/
    u_char  mincdd_year;        /*years since 1900*/
    u_char  mincdd_month;       /*month*/
    u_char  mincdd_day;         /*day*/
    u_char  mincdd_hour;        /*hour*/
    u_char  mincdd_minute;      /*minute*/
    u_char  mincdd_second;      /*second*/
    char    mincdd_timezone;    /*timezone, offset from Greenwich Mean Time
                                  in number of 15 minutes intervals from
                                  -48(West) to +52(East)*/
    u_char  mincdd_flag;        /*file flags*/
    u_char  mincdd_unit_size;   /*size (in logic blocks) of file unit*/
    u_char  mincdd_lg_size;     /*size (in logic blocks) of interleave gap*/
    u_short mincdd_vol_seq_lsb; /*sequence num. of disc has the extent(LSB)*/
    u_short mincdd_vol_seq_msb; /*sequence num. of disc has the extent(MSB)*/
    u_char  mincdd_idlen;       /*file id length in bytes*/
    char    mincdd_file_id[KMINNAMLEN];
};
```

Fields in the volume descriptor that contain character strings are not null-terminated.

SEE ALSO

cdrom(4), cdfsdir(4), cdnode(4).

Information Processing - Volume and File Structure of CD-ROM for Information Interchange, Ref. No. ISO 9660: 1988 (E).

The Working Paper for Information Processing – Volume and File Structure of" Compact Read Only Optical Discs for Information Interchange, National Information Standards Organization [Z39].

NAME

cdfsdir - format of CDFS directories

SYNOPSIS

```
#include <sys/types.h>
#include <sys/cdfsdir.h>
```

REMARKS

This entry describes the directory format for the CDFS file system. Refer to other *dir*(4) manual pages for information valid for other file systems.

DESCRIPTION

The CDFS file system supports ordinary files and directories. The fact that a file is a directory is indicated by a bit in the directory record for that file. The structure of a directory record (as given in `sys/cdfsdir.h`) is:

```
#define CDMINNAMLEN    1        /*Min. length of file identifier*/

struct min_cddir {
    u_char    mincdd_reclen;      /*length of directory record in bytes*/
    u_char    mincdd_xar_len;     /*length of XAR in logic blocks*/
    u_int     mincdd_loc_lsb;     /*logic block number of the extent in LSB
    u_int     mincdd_loc_msb;     /*logic block number of the extent in MSB
    u_int     mincdd_size_lsb;    /*size (in bytes) of the file section in
    u_int     mincdd_size_msb;    /*size (in bytes) of the file section in
    u_char    mincdd_year;        /*years since 1900*/
    u_char    mincdd_month;       /*month*/
    u_char    mincdd_day;         /*day*/
    u_char    mincdd_hour;        /*hour*/
    u_char    mincdd_minute;      /*minute*/
    u_char    mincdd_second;      /*second*/
    char      mincdd_timezone;    /*timezone, offset from Greenwich Mean Ti
                                  in number of 15 minutes intervals from
                                  -48(West) to -52(East)*/
    u_char    mincdd_flag;
    u_char    mincdd_unit_size;   /*size (in logic blocks) of file unit*/
    u_char    mincdd_lg_size;     /*size (in logic blocks) of interleave ga
    u_short   mincdd_vol_seq_lsb; /*sequence num. of disc has the extent(LS
    u_short   mincdd_vol_seq_msb; /*sequence num. of disc has the extent(MS
    u_char    mincdd_idlen;       /*file id length in bytes*/
    char      mincdd_file_id[CDMINNAMLEN];
};
```

The directory record includes the following information:

- Length of the extended attribute record, if any;
- Block number where the file begins;
- Size of the file;
- Date and time when file was recorded;
- Flag value specifying CD-ROM-specific attributes (such as file type);
- File's file unit size (for interleaving);
- File's interleave gap size (for interleaving);
- File name.

The first two records in any directory are those for `.` and `...` The first is an entry for the directory itself; the second is for the parent directory. In the case of the root directory, `.` and `..` both refer to the root directory.

Each file or directory can optionally have additional information specified for it through a data structure called the **extended attribute record** (XAR). An XAR looks like this:

Fields in this structure are defined as follows:

mincdd_reclen Length of this directory record, in bytes. Always 34.

mincdd_xar_len Length of the extended attribute record (XAR) for the root directory, if any, in logical blocks. If there is no XAR, this value is zero.

mincdd_loc_lsb Logical block number of the block in which the data for the root directory begins, in least-significant-byte-first order. If **mincdd_xar_len** is non-zero, this block number is the block where the XAR begins; otherwise, it is the block where the root directory begins.

mincdd_loc_msb Same as **mincdd_loc_lsb** above, only written in most-significant-byte-first order.

mincdd_size_lsb Length of the root directory data (excluding any XAR) in bytes, written in least-significant-byte-first order.

mincdd_size_msb Same as **mincdd_size_lsb** above, except written in most-significant-byte-first order.

mincdd_year Numerical value giving the number of years since 1900, specifying the year in which this directory record was recorded.

mincdd_month Numerical value giving the month (1 = January) in which this directory record was recorded.

mincdd_day Numerical value giving the day of the month in which this directory record was recorded.

mincdd_hour Numerical value giving the hour of the day (in 24-hour clock time) in which this directory record was recorded.

mincdd_minute Numerical value giving the minute of the hour in which this directory record was recorded.

mincdd_second Numerical value giving the second of the minute in which this directory record was recorded.

mincdd_timezone Numerical value giving the offset from Greenwich Mean Time in 15-minute intervals (–48 to 52) of the timezone in which this directory record was recorded.

mincdd_flag File flags for the root directory.

mincdd_unit_size Size of the file unit if the file is interleaved (directories cannot be interleaved, so this value is always zero for the root directory).

mincdd_lg_size Size of the interleave gap if the file is interleaved (this field is always zero for the root directory).

mincdd_vol_seq_lsb
Volume sequence number of the volume in the volume set containing the record for the root directory. This value is written in least-significant-byte-first order.

mincdd_vol_seq_msb
Same as **mincdd_vol_seq_lsb** except written in most-significant-byte-first order.

mincdd_idlen Length of the name of this file. Since the "name" of the root directory is a constant byte value of zero, this value is always one (1).

mincdd_file_id A character string giving the name of the file. The root directory's "name" is always a constant byte value of zero.

Other fields are defined as follows:

cdf_vol_set_id Character string giving the identification of the volume set of which this volume is a member. Up to 128 characters can be specified. ID is left-

justified in the field, and padded on the right with spaces.

`cdf_pb_id` Identification of the entity which specified what would be recorded on the volume set or volume group of which this volume is a member. Up to 128 characters can be specified. In ISO format only, if the first character in the field is an underscore, the remaining characters specify a file whose contents provide the identification. In both formats, the ID is left-justified in the field and padded on the right with spaces.

`cdf_pp_id` Identification of the entity which controls the preparation of the data recorded on the volume. Up to 128 characters can be specified. In ISO format only, if the first character in the field is an underscore, the remaining characters specify a file whose contents provide the identification. In both formats, the ID is left-justified in the field and padded on the right with spaces.

`cdf_ap_id` Identification of the specification for how the data is recorded on the volume. Up to 128 characters can be specified. In ISO format only, if the first character in the field is an underscore, the remaining characters specify a file whose contents provide the identification. In both formats, the ID is left-justified in the field and padded on the right with spaces.

`cdf_copyright` Specifies the name of a file in the root directory that contains the copyright statement applicable to this volume and all preceding volumes in the volume set. In ISO format, up to 37 characters can be specified; in HSG, up to 32 are possible. The filename is left-justified in the field and padded on the right with spaces.

`cdf_abstract` Specifies the name of a file in the root directory containing the abstract statement applicable to this volume. In ISO format, up to 37 characters can be specified; in HSG, up to 32 are possible. The filename is left-justified in the field, and padded on the right with spaces.

`cdf_bibliographic` Defined in ISO format only. Specifies the name of a file in the root directory which contains bibliographic records interpreted according to standards agreed upon by the originator and the recipient of the volume. Up to 37 characters can be specified. The filename is left-justified in the field and padded on the right with spaces.

Date/Time fields The next 28 fields (in HSG) or 32 fields (in ISO) specify various dates and times. Note that all fields in these dates and times contain ASCII digits (except for the `timezone` fields in ISO, which contain an eight-bit two's complement value). The creation time gives the date and time at which the volume was created (recorded). The modification time gives the date and time when the contents of the volume were last modified. The expiration time gives the date and time after which the data on the volume is no longer valid. The effective time gives the date and time after which the data becomes valid.

`cdfs_fs_version` Specifies the version number of the specification of the directory and path table records. For both formats, this value is one (1).

`cdf_appl_use` A 512-byte space whose contents are not specified by the standards, but which can be used by an application for purposes agreed upon prior to disc mastering.

`cdf_future_use` Reserved for future use; initialized to contain zeros. This field contains 653 bytes in ISO format, and 680 bytes in HSG.

NOTES

At the present time, only the `statfs()` system call (see *statfs*(2)) returns information from the volume descriptor. Information returned is limited to the logical block size and the size of the volume. The rest of the volume descriptor can only be read by means of a raw read of the CD-ROM itself.

```
#define YEAR_DIGIT      4
#define MONTH_DIGIT     2
#define DAY_DIGIT       2
#define HOUR_DIGIT      2
#define MINUTE_DIGIT    2
#define SECOND_DIGIT    2
#define ZONE_DIGIT      1

struct cdxar_iso {
    u_short  xar_uid_lsb;
    u_short  xar_uid_msb;
    u_short  xar_gid_lsb;
    u_short  xar_gid_msb;
    u_short  xar_perm;
    char     xar_create_year[YEAR_DIGIT];
    char     xar_create_month[MONTH_DIGIT];
    char     xar_create_day[DAY_DIGIT];
    char     xar_create_hour[HOUR_DIGIT];
    char     xar_create_minute[MINUTE_DIGIT];
    char     xar_create_second[SECOND_DIGIT];
    char     xar_create_centsecond[SECOND_DIGIT];
    char     xar_create_zone[ZONE_DIGIT];
    char     xar_mod_year[YEAR_DIGIT];
    char     xar_mod_month[MONTH_DIGIT];
    char     xar_mod_day[DAY_DIGIT];
    char     xar_mod_hour[HOUR_DIGIT];
    char     xar_mod_minute[MINUTE_DIGIT];
    char     xar_mod_second[SECOND_DIGIT];
    char     xar_mod_centsecond[SECOND_DIGIT];
    char     xar_mod_zone[ZONE_DIGIT];
    char     xar_exp_year[YEAR_DIGIT];
    char     xar_exp_month[MONTH_DIGIT];
    char     xar_exp_day[DAY_DIGIT];
    char     xar_exp_hour[HOUR_DIGIT];
    char     xar_exp_minute[MINUTE_DIGIT];
    char     xar_exp_second[SECOND_DIGIT];
    char     xar_exp_centsecond[SECOND_DIGIT];
    char     xar_exp_zone[ZONE_DIGIT];
    char     xar_eff_year[YEAR_DIGIT];
    char     xar_eff_month[MONTH_DIGIT];
    char     xar_eff_day[DAY_DIGIT];
    char     xar_eff_hour[HOUR_DIGIT];
    char     xar_eff_minute[MINUTE_DIGIT];
    char     xar_eff_second[SECOND_DIGIT];
    char     xar_eff_centsecond[SECOND_DIGIT];
    char     xar_eff_zone[ZONE_DIGIT];
    /*actually longer. */
};
```

The XAR contains the following information:

- User ID of the file's owner;
- Group ID of the group to which the file belongs;
- A 16-bit value specifying access permissions;
- File creation date and time;
- File's modification date and time;
- File's expiration date and time;

- File's effective date and time;
- Other system- and application-dependent data.

Refer to *cdrom*(4) for more information regarding XARs.

FILES

`/usr/include/sys/cdfsdir.h`

SEE ALSO

fsctl(2), stat(2), cdrom(4), cdfs(4), cdnode(4).

NAME

cdnode - format of a CDFS cdnode

SYNOPSIS

#include <sys/types.h>
#include <sys/cdnode.h>

DESCRIPTION

This entry describes the cdnode structure and related concepts for the CDFS file system. Refer to other *inode*(4) manual pages for information regarding the inode structure for other file systems.

The CDFS file system does not have the concept of a separate entity called an inode. The information normally found in an HFS inode is kept in a **cdnode** data structure. However, the cdnode data structure does not reside on the physical media, but instead is kept in kernel memory space only. The cdnode information is used to uniquely identify a file.

The information kept in the cdnode structure is obtained from two other data structures in the CDFS file system:

1. Directory record for the file or directory, and
2. Extended attribute record (XAR) for the file or directory, if one exists.

Because few files usually have XARs associated with them, the cdnode information most often consists only of attributes given by the directory record for the file.

Since cdnodes are kept in kernel memory, they cannot be directly accessed by the user. The `stat()` system call attempts to map whatever information is included in the cdnode for a given file into the standard stat structure (see *stat*(2)). However, since a cdnode includes information that does not have corresponding fields in the stat structure, that information cannot be mapped and therefore cannot be accessed. No method is provided to access an entire cdnode structure.

FILES

`/usr/include/sys/cdnode.h`
`/usr/include/sys/cdfsdir.h`

SEE ALSO

stat(2), cdrom(4), cdfsdir(4).

NAME

cdrom - CD-ROM background information

DESCRIPTION

The purpose of this manual entry is to provide background information pertaining to CD-ROM. Information regarding existing standards, terminology, data layout, and levels of support is given. More detailed information is available in the standard documents listed later.

Note that several topics are discussed here which are not supported in the current HP-UX release. However, these topics are included because they are useful for understanding CD-ROM formats and terminology. Refer to the DEPENDENCIES section for details regarding what items are supported in the current release.

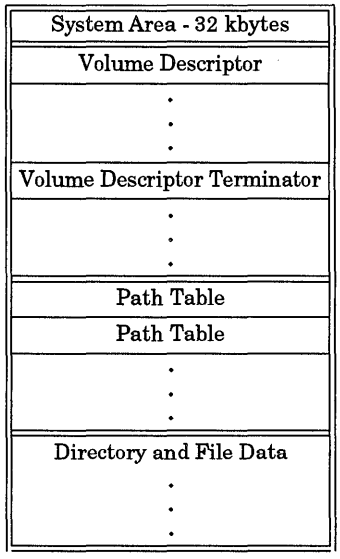
Standard Formats

Currently, there are two standard formats defined for CD-ROM. One was produced by the CD-ROM Ad Hoc Advisory Committee (popularly called the High Sierra Group, abbreviated HSG). The standard document produced by this group is called *The Working Paper for Information Processing – Volume and File Structure of Compact Read Only Optical Discs for Information Interchange*. This document is available from the National Information Standards Organization (NISO).

The second standard evolved from the HSG standard and was produced by the International Organization for Standardization (ISO). The name of the standard document is *Information Processing – Volume and File Structure of CD-ROM for Information Interchange*, reference number ISO 9660: 1988 (E).

Data Layout

The overall data layout on a CD-ROM can be represented as follows:



There are typically four sections (indicated by double horizontal lines in the table above), only two of which must occur in the order shown above. The **system area** section consists of the first sixteen 2048-byte blocks on the media. Its content is not specified by either standard, so it is possible for the creator of the CD-ROM to put data there that would be useful to the system for which the CD-ROM is intended.

The **volume descriptor** section typically contains one primary volume descriptor and zero or more supplementary volume descriptors. Each volume descriptor is 2048 bytes in length, and describes the attributes and structure of a directory hierarchy on the CD-ROM. The list of volume descriptors is terminated by one or more **volume descriptor terminators**. A volume descriptor terminator is also 2048 bytes in length, and simply signals the end of the volume descriptor section.

The **path table** section contains all the path tables for all directory hierarchies on the CD-ROM. However, path tables do not have to be placed together in this manner. They can be spaced out across the CD-ROM in whatever manner is acceptable to the person preparing data for the CD-ROM. This is often done to minimize

Directories must always consist of a single section.

Refer to *cdfsdir*(4) for more information.

Implementation and Interchange Levels

CD-ROM standards define two levels of implementation and three levels of interchange. **implementation levels** provide a way for receiving systems that support CD-ROM to specify their level of support. The implementation levels are:

Level 1 The system is permitted to ignore supplementary volume descriptors, their associated path tables, and all directory and file data associated with them.

Level 2 No restrictions apply.

In all cases, receiving systems must fulfill the receiving system requirements specified in section 10 of the ISO standard (no equivalent section exists for HSG).

Interchange levels provide a way to specify the data structure and complexity that exists on a CD-ROM. The levels are:

Level 1 Each file consists of a single file section. Filenames contain no more than eight characters, and filename extensions contain no more than three. Directory names contain no more than eight characters.

Level 2 Each file consists of a single file section.

Level 3 No restrictions apply.

DEPENDENCIES

HP-UX supports only the primary volume descriptor. When a volume is mounted, HP-UX mounts the directory hierarchy described by the first primary volume descriptor it finds. Supplementary volume descriptors are recognized and ignored, as are their associated directory hierarchies.

Directory hierarchies spanning multiple volumes are not supported.

Volume sets consisting of more than one volume are not supported.

Path tables are ignored in HP-UX. The normal pathname lookup scheme used in HFS file systems is used instead. This is done to allow other mountable file systems to be mounted on top of a mounted CDFS file system. Also, since HP-UX maintains a cache of cdnodes for CDFS files (see *cdnode*(4)), the additional performance gains provided by path tables are minimal.

HP-UX does not support multiple file sections. Each file must be recorded in a single file section.

HP-UX supports level 1 implementation and

SEE ALSO

fsctl(2), stat(2), cdfsdir(4), cdfs(4), cdnode(4).

Information Processing - Volume and File Structure of CD-ROM for Information Interchange, Ref. No. ISO 9660: 1988 (E).

The Working Paper for Information Processing – Volume and File Structure of Compact Read Only Optical Discs for Information Interchange, National Information Standards Organization [Z39].

NAME

charmap - symbolic translation file for localedef scripts

SYNOPSIS

`localedef -f` *charmap* `locale_name`

DESCRIPTION

Invoking the `localedef` command with the `-f` option causes symbolic names in the **localedef script** to be translated into the encodings given in the *charmap* file (see *localedef*(1M). A localedef script can be written partly or completely in terms of the symbolic names.

The *charmap* file has two sections: a declarations section, and a character definition section.

Declarations Section

The following declarations can precede the character definitions. Each consists of the symbol shown in the following list, including the surrounding angle brackets, followed by one or more blanks (tab or space characters), followed by the value of the symbol. No declarations are required (all are optional).

`<code_set_name>`
The name of the coded character set for which the charmap file is defined.

`<mb_cur_max>`
The maximum number of bytes in a multibyte character. Defaults to 1 if not given.

`<mb_cur_min>`
The minimum number of bytes in a character for the encoded character set. The value must be less than or equal to `<mb_cur_max>`. If not given, the default is equal to `<mb_cur_max>`.

`<escape_char>`
The character used to escape characters that otherwise would have special meaning. If not given, the default is backslash (\).

`<comment_char>`
The character used to begin comments when placed in column one of the *charmap* file. If not given, the default is the # character.

Character Definition Section

The character-set mapping definitions are the lines immediately following an identifier line containing the string `CHARMAP` and preceding a trailer line consisting of the string `END CHARMAP`. Empty lines and lines beginning with the comment character are ignored. The character definition lines are of two forms.

<symbolic_name> encoding [*comment_text*]
<symbolic_name>... <symbolic_name> encoding [*comment_text*]

The first form defines a single character and its encoding. A symbolic name is one or more visible characters from the character set illustrated in the EXAMPLES section below enclosed in angle brackets. Metacharacters such as angle brackets, escape characters, or comment characters must be escaped if they are used in the name. Two or more symbolic names can be given for the same encoding. The encoding is a character constant in one of four forms.

character	A single character has the value of that character's encoding in the current character set (i.e. the character set in the executing environment).
decimal	An escape character followed by the letter `d`, followed by one to three decimal digits.
octal	An escape character followed by one to three octal digits.
hexadecimal	An escape character followed by an `x`, followed by two hexadecimal digits.

Multibyte characters are represented by the concatenation of character constants. All constants used in the encoding of a multibyte character must be of the same form.

The second form of character definition line defines a range of characters consisting of all characters from the first symbolic name to the second, inclusive. The symbolic name must consist of one or more nonnumeric characters followed by an integer formed of one or more decimal digits. The integer part of the second symbolic name must be larger than that of the first. The range is then interpreted as a list of symbolic names consisting of the same character portion and successive integer values from the first through the last. These names are assigned successive encodings starting with the one given.

seek times.

The **directory and file data** section contains all the directory and file data for all directory hierarchies on the CD-ROM. Data can be made non-contiguous by occasional placement of a path table in the midst.

Volumes and Directory Hierarchies

A **volume** is a single physical CD-ROM. A **directory hierarchy** is a hierarchical file system written on a volume. Multiple directory hierarchies can be placed on a single volume, or a single directory hierarchy can span multiple volumes. Each directory hierarchy on a volume is described by a **volume descriptor**.

Directory hierarchies on the same volume can be totally independent of each other with each one defining a totally unique and unrelated file system. They can also be related to each other through the sharing of data between them.

A **volume set** is a set of one or more volumes that are to be treated as a unit. Each successive volume in the volume set updates or augments the data on the volumes preceding it. Thus, the last volume in a volume set is always the volume which describes the most up-to-date directory hierarchy for the volume set. A unique and ascending value called the **volume sequence number**, is assigned to each volume in a volume set. Volume sets are useful for updating large multi-volume databases without having to rework the entire set.

Volume Descriptors

Each directory hierarchy on a volume is described by a **volume descriptor**. There are several types of volume descriptors, but the two of most interest are the **primary volume descriptor** and the **supplementary volume descriptor**. Their content is almost identical, but they have different intended uses.

The primary volume descriptor describes the primary directory hierarchy on a volume. If there are additional directory hierarchies on the volume, or different ways to view the same directory hierarchy, these are described by supplementary volume descriptors. In the case of a volume set, the primary volume descriptor on each volume describes the primary directory hierarchy for that volume and all preceding volumes in the set thus far.

Volume descriptors contain the following information:

standard ID (identifies the format of the volume);
system ID;
volume ID;
size of the volume;
volume set size;
volume sequence number;
logical block size;
path table size;
pointers to the path tables;
directory record for the root directory;
volume set ID;
publisher ID;
data preparer ID;
application ID;
copyright filename;
abstract filename;
bibliographic filename (ISO only);
volume creation date and time;
volume modification date and time;
volume expiration date and time;
volume effective date and time;
application use area.

Refer to *cdfs*(4) for more detailed information about volume descriptors.

Path Tables

A **path table** defines a directory hierarchy structure within a volume. Each path table contains a record for each directory in the hierarchy. In each record are kept the directory's name, the length of any extended attribute record associated with the directory, the logical block number of the block in which the directory begins, and the number of the parent directory for that directory. (All directories in a path table are

numbered according to the order in which they appear in the path table.)

There are two types of path tables. One is a **type-L** path table in which all numerical values in each path table record are recorded least-significant-byte-first. The other type, **type-M**, is a path table in which all numerical values are recorded most-significant-byte-first. One of each type of path table is required by both standards. The ISO standard allows for one additional optional copy of each type of path table, while the HSG standard allows for up to three additional optional copies of each type. Additional copies of path tables are useful for redundancy or seek time minimization.

Extended Attribute Records

An **extended attribute record** (abbreviated **XAR**) is a data structure specifying additional information about the file or directory with which the XAR is associated. An XAR contains the following information:

owner id;
group id;
permissions;
creation date and time;
modification date and time;
expiration date and time;
effective date and time;
record information;
application use area.

Refer to *cdfsdir*(4) for more information regarding the contents of an XAR.

If an XAR is recorded, the XAR is written beginning at the first block of the file or directory. The actual data for the file or directory is written beginning at the next block after the block in which the XAR ends.

Where possible, XAR information is mapped into the stat structure by the `stat()` system call (see *stat*(2)). However, many items do not map very well due to lack of appropriate fields in the stat structure for information provided by the XAR. To preserve backward compatibility of the stat structure, such information is discarded by `stat()`. The `fsctl()` system call can be used to obtain the XAR for a particular file or directory (see *fsctl*(2)).

Interleaving

For performance reasons, data in a file can be interleaved when recorded on the volume. This is accomplished by dividing the file into pieces called **file units**. The size of each file unit (in logical blocks) is called the **file unit size**. The interleaved file is then recorded onto the volume by writing a file unit, skipping one or more blocks, writing another file unit, skipping more blocks, and so on until the entire file is recorded. The number of blocks to skip between file units is called the **interleave gap size**. Blocks making up the interleave gap are available for assignment to other files.

File unit and interleave gap sizes are kept in the directory record for each file. Thus, the file unit and interleave gap sizes may change from file to file, but cannot change within the same file (unless the file is written in **sections** – see below).

Directories cannot be interleaved.

Refer to *cdfsdir*(4) for more information.

File Sections

In order to be able to share data between files, a file can be broken up into **file sections**. File sections for a particular file are not necessarily all the same size.

Each file section is treated like a separate file in that each section gets its own directory record. This implies that each file section has its own size, its own XAR, and its own unique file unit and interleave gap sizes. However, all file sections for the same file must all share the same filename. The order of the file sections in the file is determined by the order of the directory records for each section. A bit in each directory record determines whether or not that record is the last record for the file.

A file section can appear more than once in a single file, or appear many times in many different files. A file section in one volume can also be claimed by a file in a subsequent volume in a volume set (this is how updates are accomplished).

Each file section can have its own XAR. However, if the final file section of a file has no associated XAR, the entire file is treated as if it has no XAR. This is done to make updates work sensibly.

For example, the character definition line

```
<C4>...<C6>    \d129
```

is equivalent to:

```
<C4>          \d129
<C5>          \d130
<C6>          \d131
```

EXAMPLES

The following is the charmap file for the POSIX (same as C) locale. Any charmap file is required to contain these symbolic names, but the mappings can be different for different encoded character sets.

```
<code_set_name>          ROMAN8
<mb_cur_max>             1
<mb_cur_min>             1
<escape_char>            \
<comment_char>           #

CHARMAP
<NUL>                    \000        # demonstrates octal form
<alert>                  \x07        # demonstrates hex form
<backspace>              \d8         # demonstrates decimal form
<tab>                    \011
<newline>                \d10
<vertical-tab>           \x0b
<form-feed>              \014
<carriage-return>        \d13
<space>                  \x20
<exclamation-mark>       !
<quotation-mark>         "
<number-sign>            #
<dollar-sign>            $
<percent-sign>           %
<ampersand>              &
<apostrophe>             '
<left-parenthesis>       (
<right-parenthesis>      )
<asterisk>               *
<plus-sign>              +
<comma>                  ,
<hyphen>                 -
<hyphen-minus>           -
<period>                 .
<full-stop>              .
<slash>                  /
<solidus>                /        # note duplicate definition
<zero>                   0
<one>                    1
<two>                    2
<three>                  3
<four>                   4
<five>                   5
<six>                    6
<seven>                  7
<eight>                  8
<nine>                   9
<colon>                  :
<semicolon>              ;
<less-than-sign>         <
<equals-sign>            \=
```

```
<greater-than-sign>     >
<question-mark>         ?
<commercial-at>         @
<commercial-at>         @
<A>                     A
<B>                     B
<C>                     C
<D>                     D
<E>                     E
<F>                     F
<G>                     G
<H>                     H
<I>                     I
<J>                     J
<K>                     K
<L>                     L
<M>                     M
<N>                     N
<O>                     O
<P>                     P
<Q>                     Q
<R>                     R
<S>                     S
<T>                     T
<U>                     U
<V>                     V
<W>                     W
<X>                     X
<Y>                     Y
<Z>                     Z
<left-square-bracket>   [
<backslash>             \
<reverse-solidus>       \       # note duplicate definition
<right-square-bracket>  ]
<circumflex>            ^
<circumflex-accent>     ^       # note duplicate definition
<underscore>            \_
<low-line>              \_      # note duplicate definition
<grave-accent>          `
<a>                     a
<b>                     b
<c>                     c
<d>                     d
<e>                     e
<f>                     f
<g>                     g
<h>                     h
<i>                     i
<j>                     j
<k>                     k
<l>                     l
<m>                     m
<n>                     n
<o>                     o
<p>                     p
<q>                     q
<r>                     r
<s>                     s
<t>                     t
```

```
        <u>                     u
        <v>                     v
        <w>                     w
        <x>                     x
        <y>                     y
        <z>                     z
        <left-brace>            {
        <left-curly-bracket>    {       # note duplicate definition
        <vertical-line>         |
        <right-brace>           }
        <right-curly-bracket>   }       # note duplicate definition
        <tilde>                 ~
        END CHARMAP
```

SEE ALSO

localedef(1M), localedef(4)

STANDARDS CONFORMANCE

`localedef` POSIX.2, XPG4.

NAME

checklist - static information about the file systems

SYNOPSIS

```
#include <checklist.h>
```

DESCRIPTION

checklist is an ASCII file that resides in directory **/etc**. It is only read by programs, and not written. It is the duty of the system administrator to properly create and maintain this file. **/etc/checklist** contains a list of mountable file system entries. The fields within each entry of a file system are separated by one or more blanks. Each file system entry is contained on a separate line. The order of entries in **/etc/checklist** is only important for entries without a *pass number* field. Entries without a *pass number* are sequentially checked by **fsck** (see *fsck*(1M)) after the entries with a *pass number* have been checked.

Each file system entry must contain a *special file name* and may additionally contain all of the following fields, in order:

directory
type
options
backup frequency
pass number (on parallel fsck)
comment

If any of the fields after the name of the special file are present, they must all be present in the order indicated to ensure correct placeholding.

Entries from this file are accessed by use of **getmntent()** (see *getmntent*(3X)).

The fields are separated by white space, and a **#** as the first non-whitespace character in an entry or field indicates a comment.

special file name
: is a block special file name. This field is used by **fsck**, **mount**, **swapon**, and other commands.

directory
: is the name of the root of the mounted file system that corresponds to the *special file name*. If *type* is **swapfs**, *directory* can be the name of any directory within a file system. Only one directory should be specified per file system. *directory* must already exist and must be given as an absolute path name.

type
: can be **hfs**, **cdfs**, **nfs**, **swap**, **swapfs**, or **ignore**. If *type* is **hfs**, a local HFS file system is implied. If *type* is **cdfs**, a local CD-ROM file system is implied. If *type* is **nfs**, a remote NFS file system is implied (see NETWORKING FEATURES below). If *type* is **swap**, the *special file name* is made available as an area of swap space by the **swapon** command (see *swapon*(1M)). The *options* field is valid. The fields *directory*, *pass number*, and *backup frequency* are ignored for **swap** entries. If *type* is **swapfs**, the file system in which *directory* resides is made available as swap space by **swapon**. The *options* field is valid. The fields *special file name*, *pass number*, and *backup frequency* are ignored for **swapfs** entries. Entries marked by the *type* **ignore** are ignored by all commands and can be used to mark unused sections. If *type* is specified as either **ignore**, **swap**, or **swapfs**, the entry is ignored by the **mount** and **fsck** commands (see **mount (1M)** and *fsck*(1M)). *fsck* also ignores entries with *type* specified as **cdfs** or **nfs**.

options
: appear in this entry as a comma-separated list of option keywords as found in *mount*(1M) or *swapon(1M)*. Which keywords are used depends on the parameter specified in *type*.

backup frequency
: is reserved for possible use by future backup utilities.

pass number
: is used by the **fsck** command to determine the order in which file system checks are done. The root file system should be specified with a *pass number* of 1, and other file systems should have larger numbers. File systems within a drive should have distinct numbers, but file systems on different drives can be checked on the same pass to utilize possible parallelism available in the hardware. A file system with a *pass number* of zero is ignored

NAME

collate8 - collating sequence table for languages with 8-bit character sets

DESCRIPTION

There are four language dependent collation algorithms for European languages. These algorithms are:

Two_to_one conversions:
Some languages such as Spanish require two adjacent characters to occupy one position in the collating sequence. Examples are "CH" (which follows "C") and "LL" (which follows "L").

One_to_two conversions:
Some languages such as German require one character (e.g. "sharp S") to occupy two adjacent positions in the collating sequence.

Don't-care characters:
Some languages designate certain characters to be ignored in character comparisons. For example, if - is a "don't-care" character, the strings **REACT** and **RE-ACT** would equal each other when compared.

Case and accent priority:
Many languages require a "two-pass" collating algorithm: in pass one, the accents are stripped off the letters and the resulting two strings are compared; if they are equal, a second pass with the accents replaced is performed to break the tie. Uppercase/lowercase differentiation of letters can also be handled in this fashion.

Table Description

The collating-sequence table has four sections: a file header, a sequence table, a two_to_one mapping table, and a one_to_two mapping table.

File Header:

The file header has the following format:

```
struct header {
    short int  table_len;        /* Table length */
    short int  lang_id;          /* Language id number */
    short int  reserved1;        /* Reserved */
    short int  seq_tab;          /* Address of sequence table */
    short int  seq_len;          /* Length of sequence table */
    short int  two_to_one;       /* Address of two_to_one table */
    short int  two_to_one_len;   /* Length of two_to_one table */
    short int  one_to_two;       /* Address of one_to_two table */
    short int  one_to_two_len;   /* Length of one_to_two table */
    char       low_char;         /* Lowest character */
    char       high_char;        /* Highest character */
}
```

Sequence Table:

Sequence table entries have the following format:

```
struct seq_ent {
    unsigned char   seq_no;      /* Sequence number */
    unsigned char   type_info;   /* Character type  */
}
```

The byte value of a given character is used as an index into the sequence table. The first two bits of **type_info** are used to keep track of the character type. A value zero means the character is a one_to_one character, and the other six bits in **type_info** contain its priority. A value of one or two means that **type_info** contains an index value into either the two_to_one or the one_to_two mapping table respectively. A value zero in **seq_no** means the character is a "don't care" character.

Mapping Table for two_to_one Mapped Characters:

Entries in the two_to_one table have the following format:

```
struct two_to_one {
    char   reserved1; /* Reserved */
```

```
        char   legal_char;      /* Legal character */
        struct seq_ent    seq2; /* Sequence entry for this pair */
    }
```

"Legal" two_to_one characters are listed for each particular character. "Legal" means that the combination of two characters is treated as a single character. If a match is found, the corresponding sequence entry is used for the two. Whenever a legal successor is not found in table, the character is treated according to one_to_one mapping, and the priority in the last entry combined with sequence number of the character creates the sequence entry.

Mapping Table for one_to_two Mapped Characters:
Entries in the one_to_two mapping table have the same format as entries in the sequence table. The sequence number of the first character is known from the entry in the sequence table. The sequence number of the second character is found in the one_to_two mapping entry, and the priority is used for both characters.

WARNING

This file is provided for historical reasons only. The recommended interface for native language support collation is the routines `nl_strcmp()` and `nl_strncmp()` (see *string*(3C)).

AUTHOR

`collate8` was developed by the Hewlett-Packard Company.

SEE ALSO

sort(1), nl_string(3C).

by the **fsck** command. If *pass number* is not present, **fsck** checks each such file system sequentially after all eligible file systems with pass numbers have been checked.

comment is an optional field that starts with a **#** character and ends with a new-line character. Space from the *pass number* up to the *comment* field, if present, or the new-line is reserved for future use.

There is no limit to the number of *special file name* fields in **/etc/checklist**.

NETWORKING FEATURES

NFS

If the field *type* is **nfs**, a remote NFS file system is implied. For NFS file systems, the *special file name* should be the serving machine name followed by ":" followed by the path on the serving machine of the directory to be served. The *pass number*, and *backup frequency* fields are ignored for NFS entries.

EXAMPLES

Examples of typical **/etc/checklist** entries:

Add an HFS file system at **/users** using default mount options; (backup frequency 0) fsck pass 2:

```
/dev/dsk/c0d1s0 /users hfs  defaults 0 2 # /users disk
```

Add a **swap** device with default options (*directory* field (**/**) cannot be empty, even though it is ignored):

```
/dev/dsk/c0d1s0 / swap defaults 0 0 # swap device
```

Add a swap device on a Series 300, 400, or 700 system using the space after the end of the file system (*options*=**end**):

```
/dev/dsk/0s0 / swap end 0 0 # swap at end of device
```

Add file system swap space on the file system containing directory **/swap**. *type* is **swapfs**; set *options to* **min=10**, **lim=4500**, **res=100**, and **pri=0** (see *swapon*(1M)) for explanation of meanings). *device* field is ignored but must be non-empty:

```
default /swap swapfs min=10,lim=4500,res=100,pri=0 0 0
```

(Note that both a file system entry and a swap entry are required for devices providing both services.)

DEPENDENCIES

NFS

Here is an example for mounting an NFS file system for systems that support NFS file systems:

```
server:/mnt /mnt nfs rw,hard 0 0 #mount from server.
```

AUTHOR

checklist was developed by HP, AT&T, Sun Microsystems, Inc., and the University of California, Berkeley.

SEE ALSO

fsck(1M), mount(1M), swapon(1M), getfsent(3X), getmntent(3X), mnttab(4).

NAME

clusterconf - HP Cluster configuration file, cluster.h

SYNOPSIS

```
#include <sys/types.h>
#include <cluster.h>
```

DESCRIPTION

The file `/etc/clusterconf` describes the membership of an HP cluster and is used by several library routines. The file itself has the following format:

Lines starting with `#` are comment lines.

The first non-comment line should contain the link-level address of the cluster server followed by a colon (`:`) character.

A description of each cluster node (detailed below).

A cluster node is described by a series of colon (`:`) separated fields, terminated by a new-line character. The fields are:

machine ID The ETHERNET address of the attached LAN card. This is a 12-character hexadecimal number.

cnode ID An integer between 1 and 255 inclusive. Used to identify cnodes within a cluster. Each entry in `/etc/clusterconf` must have a unique cnode ID. By convention, the cnode ID of the cluster server is 1.

cnode name The name associated with this cnode of the cluster. The cnode name can be up to 8 characters long. Each entry in `/etc/clusterconf` must have a unique cnode name.

cnode type A single character. If this machine is the cluster server, the cnode type is `r`; otherwise, it is `c`.

swap location If this machine uses the cluster server's swap space, *swap location* is the cnode ID of the cluster server. If swapping locally, *swap location* is the cnode ID of the client machine. If swapping through an auxiliary swap server, *swap location* is the cnode ID of the auxiliary swap server.

csp The default number of kernel-level server processes to create when the `csp` command is executed (see *csp*(1M)). The number of CSPs allowed on the system at any given time is limited by the value of the HP-UX tuneable parameter `ngcsp`.

The file `/etc/clusterconf` is usually accessed by the routines `getccent()`, `getccmid()`, `getccnam()`, `setccent()`, `endccent()`, and `fgetccent()` (see *getccent*(3C)).

The `cct_entry` structure defined in `<cluster.h>` is defined as follows:

```
struct cct_entry {                  /* cluster configuration table entry */
    u_char machine_id[M_IDLEN];     /* Machine ETHERNET address */
    cnode_t cnode_id;               /* cnode ID */
    char cnode_type;                /* 'r' for cluster server,
                                       'c' for all others */
    char cnode_name[15];            /* cnode name */
    cnode_t swap_serving_cnode;     /* swap server location */
    int kcsp;                       /* default number of CSPs */
}
```

AUTHOR

`clusterconf` was developed by HP.

SEE ALSO

csp(1M), rbootd(1M), getccent(3C).

NAME

core - format of core image file

DESCRIPTION

The HP-UX system writes out a file containing a core image of a terminated process when certain signals are received (see *signal*(5) for the list of reasons). The most common causes are memory violations, illegal instructions, floating point exceptions, bus errors, and user-generated quit signals. The core image file is called `core` and is written in the process's working directory (provided it is allowed by normal access controls). A process with an effective user ID different from its real user ID does not produce a core image.

The file contains sufficient information to determine what the process was doing at the time of its termination. Core file contents consist of objects that represent different segments of a process. Each object is preceded by a `corehead` data structure, and each `corehead` data structure describes the corresponding object following it. The structure is defined in <`sys/core.h`>, and includes the following members:

```
int       type;
space_t   space;
caddr_t   addr;
size_t    len;
```

The *space* and *addr* members specify the virtual memory address in the process where the described object began. The *len* member is the length of the object in bytes.

The following possible values for *type* are defined in <`sys/core.h`>:

`CORE_DATA` Process data as it existed at the time the core image was created. This includes initialized data, uninitalized data, and the heap at the time the core image is generated.

`CORE_EXEC` A compiler-dependent data structure containing the exec data structure, the magic number of the executable file, and the command (see the declaration of the `proc_exec` structure in <`sys/core.h`>).

`CORE_FORMAT` The version number of the core format produced. This number changes with each HP-UX release where the core format itself has changed. However, it does not necessarily change with every HP-UX release. `CORE_FORMAT` can thus be easily used by core-reading tools to determine whether they are compatible with a given core image. This type is expressed by a four-byte binary integer.

`CORE_KERNEL` The null-terminated version string associated with the kernel at the time the core image was generated.

`CORE_PROC` An architecture-dependent data structure containing per-process information such as hardware register contents. See the declaration of the `proc_info` structure in <`sys/core.h`>.

`CORE_STACK` Process stack contents at the time the core image was created.

Objects dumped in a `core` image file are not arranged in any particular order. Use `corehead` information to determine the type of the object that immediately follows it.

SEE ALSO

adb(1), cdb(1), xdb(1), setuid(2), crt0(3), end(3C), signal(5).

NAME

cpio - format of cpio archive

DESCRIPTION

The *header* structure, when the `-c` option of `cpio` is not used (see *cpio*(1)), is:

```
struct {
    short  c_magic,
           c_dev;
    ushort c_ino,
           c_mode,
           c_uid,
           c_gid;
    short  c_nlink,
           c_rdev,
           c_mtime[2],
           c_namesize,
           c_filesize[2];
    char   c_name[c_namesize rounded to word];
} Hdr;
```

When the `cpio -c` option is used, the *header* information is described by:

```
sscanf(Chdr,"%6ho%6ho%6ho%6ho%6ho%6ho%6ho%6ho%11lo%6ho%11lo",
           &Hdr.c_magic,&Hdr.c_dev,&Hdr.c_ino,&Hdr.c_mode,
           &Hdr.c_uid,&Hdr.c_gid,&Hdr.c_nlink,&Hdr.c_rdev,
           &Longtime,&Hdr.c_namesize,&Longfile);
```

Longtime and *Longfile* are equivalent to `Hdr.c_mtime` and `Hdr.c_filesize`, respectively. The contents of each file are recorded together with other items describing the file. Every instance of `c_magic` contains the constant 070707 (octal). The items `c_dev` through `c_mtime` have meanings explained in *stat*(2). The length of the null-terminated path name `c_name`, including the null byte, is given by `c_namesize`.

The last record of the *archive* always contains the name `TRAILER!!!`. Directories and the trailer are recorded with `c_filesize` equal to zero.

It will not always be the case that `c_dev` and `c_ino` correspond to the results of `stat()`, but the values are always sufficient to tell whether two files in the archive are linked to each other.

When a device special file is archived by HP-UX `cpio` (using the `-x` option), `c_rdev` contains a magic constant which is dependent upon the implementation doing the writing. `H_rdev` flags the device file as an HP-UX 32-bit device specifier, and `c_filesize` contains the 32-bit device specifier (see *stat*(2)). If the `-x` option is not present, special files are not archived or restored. Non-device special files are never restored.

SEE ALSO

cpio(1), find(1), stat(2).

STANDARDS CONFORMANCE

`cpio`: XPG2, XPG3, XPG4, FIPS 151-2, POSIX.1

NAME

devices - file of driver information for insf, mksf, lssf

DESCRIPTION

The **devices** file contains a description of I/O drivers, pseudo-drivers, hardware addresses and block/character major numbers. It is created by **uxgen** (see *uxgen*(1M)). This file normally resides in the directory **/etc**.

This is an ASCII file consisting of zero or more lines where each line is terminated by a new-line character. Each line begins with a name which normally represents an I/O driver or pseudo-driver. Tokens are separated by white space.

Each parameter in the line is preceded by a keyword. All parameters are optional. The keywords are: **lu**, **address**, **b_major**, and **c_major**. representing logical unit number, hardware address, block major number, character major number, respectively. Parameters can appear in any order after the name; however, they must be directly preceded by their keyword.

The following lines represent typical entries in a **devices** file:

```
cn                                                  c_major 0
disc0   lu 0    address 28.0.0  b_major 0   c_major 4
disc0   lu 1    address 28.0.2  b_major 0   c_major 4
```

AUTHOR

devices was developed by HP.

SEE ALSO

insf(1M), mksf(1M), lssf(1M), uxgen(1M).

NAME

dialups, d_passwd - dialup security control

DESCRIPTION

dialups and **d_passwd** are used to control the dialup security feature of **login** (see *login*(1)). If **/etc/dialups** is present, the first word on each line is compared with the name of the line upon which the login is being performed (including the **/dev/**, as returned by **ttyname()** (see *ttyname*(3C)). If the login is occurring on a line found in **dialups**, dialup security is invoked. Anything after a space or tab is ignored.

When dialup security is invoked, **login** requests an additional password, and checks it against that found in **/etc/d_passwd**. The command name found in the "program to use as shell" field of **/etc/passwd** is used to select the password to be used. Each entry in **d_passwd** consists of three fields, separated by colons. The first is the command name, matching an entry in **passwd**. The second is the encrypted password to be used for dialup security for those users logging in to use that program. The third is commentary, but the second colon is required to delimit the end of the password. A null password is designated with two adjacent colons. The entry for **/bin/sh** is used if no other entry matches the command name taken from **passwd**.

FILES

/etc/dialups	dial-in tty lines
/etc/d_passwd	passwords

SEE ALSO

login(1), passwd(4).

NAME

dir - format of directories on short-name HFS file systems

SYNOPSIS

```
#include <sys/types.h>
#include <sys/dir.h>
```

REMARKS

This entry describes the System V-compatible directory format for the HFS file system. It is provided strictly for backward compatibility and compatibility with applications expecting a System V file system environment. It is not compatible with the similar but more general HFS directory format in <dirent.h>, which describes a format identical to that used in an HFS file system supporting long file names up to 255 bytes in length.

The dirent structure defined in <dirent.h> should be used in conjunction with the *directory*(3C) routines for portability to other industry UNIX implementations.

DESCRIPTION

A directory behaves exactly like an ordinary file, except that no user can write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry (see *fs*(4)). The structure of a directory entry as given in the <sys/dir.h> header file is:

```
#define DIRSIZ                14
#define DIRSIZ_CONSTANT       14
#define DIR_PADSIZE           10
#define MAXNAMLEN             255
struct      direct {
    u_long  d_ino;            /* inode number of entry */
    u_short d_reclen;         /* length of this record */
    u_short d_namlen;         /* length of string in d_name */
    char    d_name[DIRSIZ_CONSTANT];
    char    d_pad[DIR_PADSIZE];
};

/*
 * DIRSTRCTSIZ is the number of bytes in the structure
 * representing a System V-compatible (14-character
 * maximum file name length) HFS directory entry.
 */

#define DIRSTRCTSIZ   32        /* sizeof(struct direct) */
```

By convention, the first two entries in each directory are for . and .. ("dot" and "dot dot"). The first is an entry for the directory itself. The second is for the parent directory. The meaning of .. is modified for the root directory of the master file system; there is no parent, so . . and . have the same meaning.

AUTHOR

dir was developed by AT&T and HP.

SEE ALSO

fs(4), directory(3C).

NAME

disktab - disk description file

SYNOPSIS

```
#include <disktab.h>
```

DESCRIPTION

`disktab` is a simple database that describes disk geometries and disk section characteristics. Entries in `disktab` consist of a number of colon-separated fields. The first entry for each disk gives the names by which the disk is known, separated by vertical bar (|) characters. The last name given should be a long name fully identifying the disk.

The following list indicates the normal values stored for each disk entry. Sectors are of size `DEV_BSIZE`, defined in `<sys/param.h>`.

Name	Type	Description
`ns`	num	Number of sectors per track
`nt`	num	Number of tracks per cylinder
`nc`	num	Total number of cylinders on the disk
`b0`	num	Block size for section '0' (bytes)
`b1`	num	Block size for section '1' (bytes)
`b`*n*	num	Block size for section '*n*' (bytes)
`f0`	num	Fragment size for section '0' (bytes)
`f1`	num	Fragment size for section '1' (bytes)
`f`*n*	num	Fragment size for section '*n*' (bytes)
`s0`	num	Size of section '0' in sectors
`s1`	num	Size of section '1' in sectors
`s`*n*	num	Size of section '*n*' in sectors
`rm`	num	Revolution per minute
`ty`	`str`	Type of disk (e.g. removable, winchester)

Example:

```
hp7914:
    :ty=winchester:ns#16:nt#7:nc#1061
    :s0#118832
        :b0#8192:f0#1024:rm#3600:
```

DEPENDENCIES

Series 300/400/700:

There is only one section per disk drive.

FILES

`/etc/disktab`

AUTHOR

`disktab` was developed by HP and the University of California, Berkeley.

SEE ALSO

newfs(1M), getdiskbyname(3C).

NAME

DOSIF - DOS Interchange Format description

DESCRIPTION

DOSIF (DOS Interchange Format) is the name given to the media format used by the DOS operating system. This format is based upon that used in IBM PC and PC AT, HP Vectra, and HP 150 systems.

The DOS utilities described in Section 1 (referred to hereafter as *dos**(1)) are provided for reading data from and writing data to DOSIF volumes. Use these utilities to retrieve information from a DOSIF volume.

The *dos**(1) utilities are the only HP-UX commands that can interact directly with the contents of a DOSIF volume. The only other way to interact with the contents of a DOSIF volume is to use an HP-UX DOS emulation or coprocessor facility such as SoftPC or the DOS Coprocessor. **mount** cannot be used on a DOSIF volume because the operating system does not recognize it (see *mount*(1)).

When constructing file names for *dos**(1) commands, start with the HP-UX path name of the DOSIF volume, then add a colon (**:**) followed by the file name:

device_file **:** *file*

or

path_name **:** *file*

Note: This file naming convention is suitable for use only in arguments to the *dos**(1) utilities. It does not constitute a legal path name for any other use in HP-UX applications.

Note: Shell metacharacters (*****, **?**, and **[...]**) can be used to name HP-UX files, but cannot be used when specifying a DOS file name, because file name expansion is done by the shell and the *dos**(1) utilities do not recognize metacharacters.

By convention, if the HP-UX device name and a trailing colon are specified, but no file or directory name is provided (for example, **/dev/rfd.0:**), the root (**/**) of the DOS file system is assumed.

EXAMPLES

Specify DOSIF file **/dos/ivy** accessed through HP-UX special file **/dev/rfd9127**:

```
/dev/rfd9127:/dos/ivy
```

Specify DOSIF file **/math** accessed through the DOS volume stored as HP-UX file **/users/mydir/driveC**:

```
/users/mydir/driveC:/math
```

SEE ALSO

dos2ux(1), doschmod(1), doscp(1), dosdf(1), dosls(1), dosmkdir(1), dosrm(1).

NAME

dp - dedicated ports file, used by DDFA and DTC port ID

DESCRIPTION

The `dp` file has two uses:

DTC Port ID via Telnet

The `dp` file is used by the HP-UX telnet daemon (`telnetd`) to identify the calling port and board of a telnet connection from an HP Datacommunications and Terminal Controller (DTC)

At connection time, the host negotiates the telnet environment option, and the DTC replies with the port and board number of the connecting device. `telnetd` maps the port and board numbers to the well-known name for the device, which has previously been configured in the `dp` file.

DTC Device File Access (DDFA)

The `dp` file is used by the HP Datacommunications and Terminal Controller (DTC) Device File Access (DDFA) software to allow terminal server ports to be programatically accessed from HP-UX applications in the same way as devices connected directly to the HP-UX system. It contains a one-line entry for each configured DTC port.

The `dp` file contains the information the DDFA software needs to set up and manage a connection to a specified DTC port. The file is parsed by the Dedicated Port Parser (`dpp`) which spawns an outgoing connection daemon for each connection specified in the file.

Port ID via Telnet

To configure the `dp` file for use in port ID via telnet, the default file `/etc/newconfig/ddfa/dp` should be copied to a new file, and the copy configured with the appropriate values for the incoming connections. We recommend you create a directory `/etc/ddfa` to hold the `dp` file and the modified `pcf`s.

`pcf` information is in the following format:

dtc_ip_address board /port pseudonym

The exact details of each field are given in the Configuration Information section below.

DTC Outbound Connections

For DTC outbound connections, the following information is required:

dtc_ip_address board /port pseudonym config_file

The exact details of each field are given in the Configuration Information section below.

Configuration Information

There are three ways to specify the DTC port:

- Explicitly specify its IP address.
- Specify the IP address of the DTC then specify the port and board.
- Specify the IP address of the DTC and the TCP port service address of the port.

Comments can be appended by starting them with a # character; everything after the # is ignored by the parser. Fields are separated by space characters.

Refer to *ddfa*(7) for information about how to configure and install DDFA software.

The `dp` file has the following format:

dtc_ip_address This is the IP address of the DTC being accessed, or the IP address of the port on the DTC.

board /port This field contains the DTC board and port numbers, separated by the / character. It is not necessary to pad the values with leading zeros. The board and port numbers are not checked by `dpp`, but are checked by `ocd`. Valid values are 0 through 7 for *board*, and 0 through 31 for *port* (these restrictions do not apply if the TCP service port address is specified instead).

If the *dtc_ip_address* field explicitly defines the DTC port, the value in the *port /board* field must be `xx/xx` (use `X` or `x`).

If the entry is of the form **xx/***n* where *n* is a decimal number, *n* is assumed to be the TCP port address, and this value is used when the connection is established. Otherwise, the destination is filled using the DTC formula:

$$(256\times(32\times\text{board}+\text{port}+1)+23)$$

pseudonym This is the absolute path and name of the device file known to the system and/or the end-user application. The device file name is limited to 14 characters. We recommend that the name reflect the connected device, and be similar to the **pcf** name, for example **ocd_dtc1b1p1**.

pc_file_path This is the path to the Port Configuration file (**pcf**) which contains the configuration information for the DTC port. This field is mandatory because the parser **dpp** uses the presence of this field as its flag to spawn a daemon for the line. We recommend that the name of the file reflect the connected device, and be similar to the **pcf** name, for example **pcf_dtc1b1p1**.

EXAMPLES

The following examples illustrate file entry syntax.

A printer is connected to port 1 of board 3 of a DTC with the IP address 11.234.87.123. The device attached to the port can be accessed with the HP-UX spooler by using the device file **/dev/ocd_lp1** in the **lpadmin** command. The port is to be profiled using data in the file **/etc/ddfa/pcf_lp1**:

```
11.234.87.123 03/01 /dev/ocd_lp1 /etc/ddfa/pcf_lp1 # lp1 b1,n2,f7
```

Consider a printer connected to the DTC port at IP address 11.234.87.124. The *board/port* field contains **xx/xx**. The file **pcf_lp2** contains port profiling information:

```
11.234.87.124 xx/xx /dev/ocd_lp2 /etc/ddfa/pcf_lp2 # lp2 b2,n1
```

Specify a port using a TCP port address. The port address is calculated using the formula $(256\times(32\times\text{board}+\text{port}+1)+23)$:

```
11.234.87.215 xx/16919 /dev/ocd_lp3 /etc/ddfa/pcf_lp3 # lp3 b2,p1
```

Create an entry for port ID via telnet. **telnetd** uses this entry to map the DTC's port and board numbers to the name being used for the connection on the HP-UX system:

```
11.234.87.215 02/01 terminal02
```

FILES

```
/etc/dpp
/etc/ocdbug
/etc/ocd
/etc/dpp_login.bin
/etc/utmp.dfa
/etc/newconfig/ddfa/pcf
/etc/newconfig/ddfa/dp
```

SEE ALSO

ddfa(7) dpp(1m) ocd(1m) ocdebug(1m) pcf(4).

NAME

exports, xtab - directories to export to NFS clients

SYNOPSIS

/etc/exports

/etc/xtab

DESCRIPTION

File **/etc/exports** describes the directories that can be exported to NFS clients. The system administrator creates it using a text editor. **mountd** processes it each time a mount request is received (see *mountd*(1M)).

/etc/exports is read automatically by the **exportfs** command (see **exportfs (1M)**). If this file is changed, **exportfs** must be run (**exportfs -a**) before the changes can affect the daemon's operation.

Only when this file is present at boot time does the **/etc/netnfsrc** script execute **exportfs** and the NFS filesystem daemon, **nfsd** (see *nfsd(1M)*).

/etc/xtab contains entries for directories that are currently exported. This file should only be accessed by programs using **getexportent** (see *exportent*(3)). (Use **exportfs -u** to remove entries from this file).

An entry for a directory consists of a line of the following form:

directory -option[**,***option*]...

Where *directory* is the pathname of a directory (or file).

options can have any of the following values and forms:

ro Export the directory read-only. If not specified, the directory is exported read-write.

rw=*hostname*[**:***hostname*]...
Export the directory read-mostly. Read-mostly means read-only to most machines, but read-write to those specified. If not specified, the directory is exported read-write to all.

anon=*uid*
If a request comes from an unknown user, use *uid* as the effective user ID. **Note:** Root users (uid 0) are always considered "unknown" by the NFS server unless they are included in the "root" option below.

The default value for this option is 65 534. Setting **anon** to 65 535 disables anonymous access.

root=*hostname*[**:***hostname*]...
Give root access only to the root users from a specified hostname. The default is for no hosts to be granted root access.

access=*client*[**:***client*]...
Give mount access to each client listed. A client can either be a hostname or a netgroup (see *netgroup*(4)). Each client in the list is first checked in the netgroup database, then in the hosts database. A directory name with no accompanying name list allows any machine to mount the given directory.

async Specifying **async** increases write performance on the NFS server by causing asynchronous writes on the NFS server. The async option can be specified anywhere on the command line after the file system name. Before using this option, refer to WARNINGS below.

A # character anywhere in the file indicates a comment that extends to the end of the line.

/etc/exports contains a list of file systems and the **netgroup** or machine names allowed to remotely mount each file system (see *netgroup*(4)). The file system names are left-justified and followed by a list of names separated by white space. The names are searched for in **/etc/netgroup** then in **/etc/hosts**. A file system name with no accompanying name list means the file system is available to everyone.

A # anywhere in the file indicates a comment extending to the end of that line.

EXAMPLES

```
/usr/games  cocoa fudge                 # export to only these machines
/usr        -access=clients             # export to my clients
/usr/local                              # export to the world
/usr2       -access=bison:deer:pup      # export to only these machines
/usr/adm    -root=bison:deer            # give root access only to these
/usr/new    -anon=0                     # give all machines root access
/usr/temp   -rw=ram:alligator           # export read-write only to these
/usr/bin    -ro                         # export read-only to everyone
/usr/stuff  -access=bear,anon=-65534,ro
                                        # several options on one line
```

WARNINGS

You cannot export either a parent directory or a subdirectory of an exported directory that resides *within the same filesystem*. It is not allowed, for instance, to export both `/usr` and `/usr/local` if both directories reside on the same disk partition.

Do not use the `async` option if one of the following conditions applies to a file system that you want to export:

- The file system contains files that are accessed using the `O_SYNCIO` flag (which is set by `fcntl()` or `open()` calls (see *fcntl*(2) and *open*(2)).
- The file system contains data that cannot be reconstructed (for example, the file system contains database files),
- The file system contains files synchronized with *fsync(2), or*
- The file system contains critical applications requiring absolute data integrity. If you are unsure whether any of the previous conditions apply, *do not use the async option.* If the `async` option is used, an unreported data loss may occur if the option is set and the NFS server hardware experiences a power loss, system panic, or other failure. Specifically, blocks which have been queued for the server's disk, but have not yet been written to the disk *may* be lost.

AUTHOR

`exports` was developed by Sun Microsystems, Inc.

FILES

```
/etc/exports
/etc/xtab
/etc/hosts
/etc/netgroup
/etc/netnfsrc
```

SEE ALSO

exportfs(1M), mountd(1M), nfsd(1M), exportent(3), hosts(4), netgroup(4).

NAME

fs - format of file system volume

SYNOPSIS

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/fs.h>
#include <sys/inode.h>
#include <sys/ino.h>
#include <sys/sysmacros.h>
```

DESCRIPTION

Every file system storage volume has a common format for certain vital information. The first 8 kbytes on a volume contain a volume header which identifies that volume as a Logical Interchange Format (LIF) volume. Such volume may be divided into a number of sections.

Each section can contain a file system. The first 8 kbytes in each section is ignored, except where it coincides with the volume header discussed above. The actual file system begins next with the "super block." The layout of the super block as defined by the include file <sys/fs.h> is:

```
#define FS_MAGIC        0x011954
#define FS_MAGIC_LFN    0x095014
#define FS_CLEAN        0x17
#define FS_OK           0x53
#define FS_NOTOK        0x31
struct fs {
    struct fs  *fs_link;        /* linked list of file systems */
    struct fs  *fs_rlink;       /* used for incore super blocks */
    daddr_t    fs_sblkno;       /* addr of super-block in filesys */
    daddr_t    fs_cblkno;       /* offset of cyl-block in filesys */
    daddr_t    fs_iblkno;       /* offset of inode-blocks in filesys*/
    daddr_t    fs_dblkno;       /* offset of first data after cg */
    long       fs_cgoffset;     /* cylinder group offset in cylinder*/
    long       fs_cgmask;       /* used to calc mod fs_ntrak */
    time_t     fs_time;         /* last time written */
    long       fs_size;         /* number of blocks in fs */
    long       fs_dsize;        /* number of data blocks in  fs */
    long       fs_ncg;          /* number of cylinder groups */
    long       fs_bsize;        /* size of basic blocks in fs */
    long       fs_fsize;        /* size of frag blocks in fs */
    long       fs_frag;         /* number of frags in a block in fs*/
/* these are configuration parameters */
    long       fs_minfree;      /* minimum percentage of free blocks*/
    long       fs_rotdelay;     /* num of ms for optimal next block */
    long       fs_rps;          /* disk revolutions per second */
/* these fields can be computed from the others */
    long       fs_bmask;        /* ''blkoff'' calc of blk offsets */
    long       fs_fmask;        /* ''fragoff'' calc of frag offsets */
    long       fs_bshift;       /* ''lblkno'' calc of logical blkno */
    long       fs_fshift;       /* ''numfrags'' calc number of frags*/
/* these are configuration parameters */
    long       fs_maxcontig;    /* max number of contiguous blks */
    long       fs_maxbpg;       /* max number of blks per cyl group */
/* these fields can be computed from the others */
    long       fs_fragshift;    /* block to frag shift */
    long       fs_fsbtodb;      /* fsbtodb and dbtofsb shift constant*/
    long       fs_sbsize;       /* actual size of super block */
    long       fs_csmask;       /* csum block offset */
    long       fs_csshift;      /* csum block number */
    long       fs_nindir;       /* value of NINDIR */
    long       fs_inopb;        /* value of INOPB */
```

```
    long        fs_nspf;        /* value of NSPF */
    long        fs_sparecon[6]; /* reserved for future constants */
/* sizes determined by number of cylinder groups and their sizes */
    daddr_t     fs_csaddr;      /* blk addr of cyl grp summary area */
    long        fs_cssize;      /* size of cyl grp summary area */
    long        fs_cgsize;      /* cylinder group size */
/* these fields should be derived from the hardware */
    long        fs_ntrak;       /* tracks per cylinder */
    long        fs_nsect;       /* sectors per track */
    long        fs_spc;         /* sectors per cylinder */
/* this comes from the disk driver partitioning */
    long        fs_ncyl;        /* cylinders in file system */
/* these fields can be computed from the others */
    long        fs_cpg;         /* cylinders per group */
    long        fs_ipg;         /* inodes per group */
    long        fs_fpg;         /* blocks per group * fs_frag */
/* this data must be re-computed after crashes */
    struct      csum fs_cstotal; /* cylinder summary information */
/* these fields are cleared at mount time */
    char        fs_fmod;        /* super block modified flag */
    char        fs_clean;       /* file system is clean flag */
    char        fs_ronly;       /* mounted read-only flag */
    char        fs_flags;       /* currently unused flag */
    char        fs_fsmnt[MAXMNTLEN];/* name mounted on */
/* these fields retain the current block allocation info */
    long        fs_cgrotor;     /* last cg searched */
    struct      csum  *fs_csp[MAXCSBUFS]; /* list of fs_cs info buffers */
    long        fs_cpc;         /* cyl per cycle in postbl */
    short       fs_postbl[MAXCPG][NRPOS];/*head of blocks per rotation */
    long        fs_magic;       /* magic number */
    char        fs_fname[6];    /* name of file system */
    char        fs_fpack[6];    /* pack name of file system */
    u_char      fs_rotbl[1];    /* list of blocks for each rotation */
/* actually longer */
};
```

A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A file system is described by its super-block, which in turn describes the cylinder groups. The super-block is critical data and is replicated in each cylinder group to protect against catastrophic loss. This is done at file system creation time and the critical super-block data does not change, so the copies need not be referenced further unless disaster strikes.

Addresses stored in inodes are capable of addressing fragments of 'blocks'. File system blocks of at most size **MAXBSIZE** can be optionally broken into smaller pieces, each of which is addressable; these pieces may be **DEV_BSIZE**, or some multiple of a **DEV_BSIZE** unit (**DEV_BSIZE** is defined in **<sys/param.h>**).

Large files consist of exclusively large data blocks. To avoid undue wasted disk space, the last data block of a file is allocated only as many fragments of a large block as are necessary, if that file is small enough to not require indirect data blocks. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determinable from information in the inode, using the **blksize(fs, ip, lbn)** macro.

The file system records space availability at the fragment level; to determine block availability, aligned fragments are examined.

I-numbers begin at 0. Inodes 0 and 1 are reserved. Inode 2 is used for the root directory of the file system. The **lost+found** directory is given the next available inode when it is initially created by **mkfs**.

fs_minfree gives the minimum acceptable percentage of file system blocks that can be free. If the freelist drops below this level, only the super-user may continue to allocate blocks. This can be set to 0 if no reserve of free blocks is deemed necessary. However, severe performance degradations result if the file

system is run at greater than 90% full; thus the default value of **fs_minfree** is 10%.

The best trade-off between block fragmentation and overall disk utilization and performance varies for each intended use of the file system. Suggested values can be found in the system administrator's manual for each implementation.

Cylinder-Group-Related Limits

Each cylinder keeps track of the availability of blocks at different rotational positions, so that sequential blocks can be laid out with minimum rotational latency. NRPOS is the number of rotational positions which are distinguished. For example, with NRPOS 8 the resolution of the summary information is 2ms for a typical 3600 rpm drive.

fs_rotdelay gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for **fs_rotdelay** is 2ms. Suggested values of **fs_rotdelay** for different disks can be found in the system administrator's manual.

Each file system has a statically allocated number of inodes. An inode is allocated for each **NBPI** bytes of disk space. The inode allocation strategy is extremely conservative.

MAXIPG bounds the number of inodes per cylinder group, and is needed only to keep the structure simpler by having only a single variable size element (the free bit map).

Important Note: **MAXIPG** must be a multiple of **INOPB(fs)**.

MINBSIZE is the smallest allowable block size. With a **MINBSIZE** of 4096, it is possible to create files of size 2^{32} with only two levels of indirection. **MINBSIZE** must be big enough to hold a cylinder group block, thus **MINBSIZE** must always be greater than **sizeof(struct cg)**. Note that super blocks are never more than size **SBSIZE**.

The path name on which the file system is mounted is maintained in **fs_fsmnt**. **MAXMNTLEN** defines the amount of space allocated in the super block for this name. The limit on the amount of summary information per file system is defined by **MAXCSBUFS**. It is currently parameterized for a maximum of two million cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from **fs_csaddr** (size **fs_cssize**) in addition to the super block.

Important Note: **sizeof (struct csum)** must be a power of two in order for the **fs_cs** macro to work.

The two possible values for **fs_magic** are **FS_MAGIC**, the default magic number for an HFS file system with a fixed-size directory format that limits file name length to **DIRSIZ** (14), and **FS_MAGIC_LFN**, the magic number of a file system using a variable-size directory format that supports file names of up to **MAXNAMLEN** (255) characters in length.

Super Block for a File System:

MAXBPC bounds the size of the rotational layout tables and is limited by the fact that the super block is of size **SBSIZE**. The size of these tables is inversely proportional to the block size of the file system. The size of the tables is increased when sector sizes are not powers of two, as this increases the number of cylinders included before the rotational pattern repeats (**fs_cpc**). The size of the rotational layout tables is derived from the number of bytes remaining in **(struct fs)**.

MAXBPG bounds the number of blocks of data per cylinder group, and is limited by the fact that cylinder groups are, at most, one block. The size of the free block table is derived from the size of blocks and the number of remaining bytes in the cylinder group structure **(struct cg)**.

inode:

The inode is the focus of all file activity in the HP-UX file system. There is a unique inode allocated for each active file, each continuation inode, each current directory, each mounted-on file, text file, and the root. An inode is "named" by its device-and-i-number pair. For the format of an inode and its flags, see *inode*(4).

DEPENDENCIES

Series 300/400/700

Series 300, 400, and 700 systems support only one section per volume. Thus, there can only be one file system on each volume and the first 8 Kbytes of a file system is the boot area. This area contains the LIF volume header, the directory that defines the contents of the volume, and the bootstrapping program.

AUTHOR

`fs` was developed by HP and the University of California, Berkeley.

SEE ALSO

inode(4), lif(4).

NAME

fspec - format specification in text files

DESCRIPTION

It is sometimes convenient to maintain text files on the HP-UX system with non-standard tabs, (meaning tabs that are not set at every eighth column). Generally, such files must be converted to a standard format – frequently by replacing all tabs with the appropriate number of spaces – before they can be processed by HP-UX system commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets `<:` and `:>`. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

`t`*tabs* The `t` parameter specifies tab settings for the file. The value of *tabs* must be one of the following:

1. A list of column numbers separated by commas, indicating tabs set at the specified columns;
2. A `-` followed immediately by an integer *n*, indicating tabs at intervals of *n* columns;
3. A `-` followed by the name of a "canned" tab specification.

Standard tabs are specified by `t-8`, or equivalently, `t1,9,17,25,` etc. Recognized canned tabs are defined by the `tabs` command (see *tabs*(1)).

`s`*size* The `s` parameter specifies a maximum line size. The value of *size* must be an integer. Size checking is performed after tabs have been expanded, but before the margin is inserted at the beginning of the line.

`m`*margin* The `m` parameter specifies a number of spaces to be inserted at the beginning of each line. The value of *margin* must be an integer.

`d` The `d` parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

`e` The `e` parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values (assumed for parameters not supplied) are `t-8` and `m0`. If the `s` parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

If a format specification can be disguised as a comment, it is not necessary to code the `d` parameter.

Several HP-UX system commands correctly interpret the format specification for a file. Among them is `ed`, which can be used to convert files to a standard format acceptable to other HP-UX system commands.

SEE ALSO

ed(1), newform(1), tabs(1).

NAME

fstab - static information about the file systems

SYNOPSIS

```
#include <fstab.h>
```

REMARKS

This file is provided only for portability of applications developed under BSD. New applications should use <mntent.h> and `getmntent()` (see *getmntent*(3)).

DESCRIPTION

Including <fstab.h> provides portability for BSD applications using `getfsent()` (see *getfsent*(3X)). `getfsent()` reads file `/etc/checklist` and fills in the `fstab` structure defined in <fstab.h>. Note that, in BSD systems, `/etc/fstab` is read using the *getfsent*(3X) routines. On HP-UX systems, *getfsent*(3X) emulates the BSD behavior. Note also that including and using <checklist.h> does not produce the same behavior from *getfsent*(3X) as including and using <fstab.h>.

The `struct fstab` structure is declared in <fstab.h>, and includes the following members:

```
char    *fs_spec    Block special device name
char    *fs_file    File system path prefix
char    *fs_type    Type of file system (FSTAB_*)
int     fs_passno   Pass number on parallel dump
int     fs_freq     Dump frequency, in days
```

BSD applications that depend on the position of elements within the structure are not portable.

AUTHOR

`fstab` was developed by HP, and the University of California, Berkeley.

SEE ALSO

getfsent(3X), getmntent(3X), checklist(4), mnttab(4).

NAME

ftpusers - security file for *ftpd*(1M)

DESCRIPTION

ftpd rejects remote logins to local user accounts that are named in **/etc/ftpusers**. Each restricted account name must appear alone on a line in the file. The line cannot contain any white space. User accounts that specify a restricted login shell in **/etc/passwd** should be listed in **/etc/ftpusers** because **ftpd** accesses local accounts without using their login shells. UUCP accounts should be listed in **/etc/ftpusers**. If **/etc/ftpusers** does not exist, **ftpd** skips the security check.

EXAMPLES

Given an **/etc/ftpusers** file containing the following:

```
Only lines that exactly match user account names are
significant. Blank lines are harmless because they
do not match any account names. However you must be
careful.

uucp
guest
```

ftpd would reject login attempts using the local accounts **careful.**, **uucp**, or **guest**.

AUTHOR

ftpusers was developed by the University of California, Berkeley.

SEE ALSO

ftp(1), ftpd(1M).

NAME

gated.conf - gated configuration file syntax

SYNOPSIS

`/etc/gated.conf`

DESCRIPTION

The gated config file consists of a sequence of statements terminated by a semicolon (`;`). Statements are composed of tokens separated by white space, which can be any combination of blanks, tabs and new-line characters.

Comments can be specified in either of two forms:

- Starting with a `#` character and running to the end of the line.
- "C" style, which starts with a `/*` and continues until it reaches `*/`.

Statement Classes

There are six classes of statements. The first two classes can be specified in the configuration file in any order:

Directive statements
: These statements are acted upon immediately by the parser, and are used to specify included files and the directory in which included files reside. Unlike other statements which terminate with a semicolon (`;`), directive statements terminate with a new-line character.

Trace option statements
: These statements control tracing options.

The four remaining classes must be specified in order:

Definition statements
: These statements specify options, the autonomous system, martian networks, and interface options.

Protocol statements
: These statements enable or disable protocols and set protocol options.

Static route statements
: Static routes are defined by route statements.

Control statements
: Control statements define routes that are accepted from routing peers and routes that are propagated to those peers.

Each of these statement classes are described in detail later in this manual entry.

Statement Primitives

The following primitives are used in statement definitions:

host
: Any host. A host can be specified by its IP address or by a domain name. If a domain name is specified that has multiple IP address it is considered an error. The host bits in the IP address must be non-zero.

network
: Any network. A network can be specified by its IP address or a network name. The host bits in a network specification must be zero. `default` can also be used to specify the default network (`0.0.0.0`).

destination
: Any host or network.

dest_mask
: Any host or network with an optional mask:

  `all`
  network
  network `mask` *mask*

  A mask is a dotted quad specifying which bits of the destination are significant. `all` can be used to specify that any IP address may be matched.

gateway Must be a host on an attached network.

interface Specified by IP address, domain name, or interface name. Be careful when using interface names because future UNIX operating systems may allow more than one address per interface.

gateway_list List of one or more gateways.

interface_list List of one or more interface names or addresses, or the token **all**, which refers to all interfaces. The token **all**, cannot be used with any of the control statements (such as **accept**, **propagate**, and **proto**).

preference Used to determine the order of routes to the same destination in the routing table. **gated** allows one route to a destination per protocol per autonomous system. In the case of multiple routes, the route to use is chosen by *preference*, which is a number between 0 and 255, with 0 being the most preferred and 255 being the least preferred.

In case of a preference tie, if the two routes are from the same protocol and from the same autonomous system, **gated** chooses the route with the lowest metric. Otherwise **gated** chooses the route with the lowest numeric next-hop gateway address.

metric A valid metric for the specified protocol.

Directive Statements

%directory "*path_name*"

Sets the current directory to *path_name*. This is the directory where **gated** looks for included files that do not begin with **/**.

Note that this statement does not actually change the current directory; it only specifies the prefix applied to included file names.

%include "*filename*"

Causes the specified file to be parsed completely before resuming with this file. Nesting up to 10 levels is supported.

Trace Statements

tracefile "*filename*" [**replace**] **;**

Specifies the file to contain tracing output. Trace information is appended to this file unless **replace** is specified.

traceoptions *traceoption* [*traceoption* [...]] **;**

Changes the tracing options to those specified. If **none** is the only option specified, tracing is turned off. Trace flags are:

all	Turn on all of the following options, except nostamp.
general	Turn on internal, external and route.
internal	Internal errors and informational messages.
external	External errors.
nostamp	Do not timestamp all messages in the trace file.
mark	Output a message to the trace log every 10 minutes to ensure gated is still running.
task	Task scheduling, signal handling and packet reception.
timer	Timer scheduling.
lex	Objects the lexical analyzer locates in the config file.
parse	Tokens the parser recognizes in the config file.
config	Redisplays statements read from the config file after they are parsed. This allows verification that the statements were parsed correctly.
route	Changes to the gated routing table.

`kernel`	Changes to the kernel's routing table.
`bgp`	BGP packets sent and received. May be modified by "update" and "protocol".
`egp`	EGP packets sent and received. May be modified by "update" and "protocol".
`rip`	RIP packets sent and received. May be modified by "update".
`hello`	HELLO packets sent and received. May be modified by "update".
`icmp`	ICMP redirect packets sent and received. May be modified by "update". Note that redirects processed are traced under the "route" option.
`protocol`	Provide messages about protocol state machine transitions when used with "egp" or "bgp".
`update`	Trace the contents of protocol packets.

Definition Statements

`options` *option_list* `;`

Sets `gated` options:

`noinstall` Do not change kernel's routing table. Useful for verifying configuration files.

`gendefault` BGP and EGP neighbors should cause the internal generation of a default route when up. This route is not installed in the kernel's routing table, but can be announced by other protocols. Announcement is controlled by referencing the special protocol `default`.

`autonomoussystem` *autonomous_system* `;`

Sets the autonomous system of this router to be *autonomous_system*. This option is required if BGP or EGP is being used.

`interface` *interface_list interface_options* `;`

Sets interface options on the specified interfaces. An interface list is `all` or a list of interface names (see previous warning about interface names), domain names, or numeric addresses.

Interface options are:

`metric` *metric*

Set the interface metric for this interface. This metric is used by RIP and HELLO. Specifying the metric here overrides for internal use, but does not change the metric set by `ifconfig`.

`preference` *pref*

Sets the preference for routes to this interface.

`passive`

Prevents gated from deleting the route to this interface if it is believed to be down due to routing information not received.

```
martians {
    martian_list
} ;
```

Defines a list of martian addresses about which all routing information is ignored. The <martian_list> is a semi-colon separated list of symbolic or numeric hosts with optional masks. See dest_mask.

Protocol Statements

Enables or disables use of a protocol and controls protocol options. These can be specified in any order.

For all protocols, `preference` controls the choice of routes learned via this protocol or from this autonomous system in relation to routes learned from other protocols and/or autonomous systems. The default metric used when propagating routes learned from other protocols is specified with `defaultmetric` which itself defaults to the highest valid metric for this protocol; for many protocols this signifies a lack of reachability.

For distance vector IGPs (RIP and HELLO) and redirects (ICMP), the **trustedgateways** clause supplies a list of gateways providing valid routing information; routing packets from others are ignored. This defaults to all gateways on the attached networks. In addition to routing packets to the remote end of point-to-point (**pointopoint**) links and the broadcast address of broadcast-capable interfaces, routing updates can be sent to specific gateways if they are listed in a **sourcegateways** clause and **pointopoint** or **supplier** is specified. Disabling the transmission and reception of routing packets for the protocols can be specified with the **interface** clause.

For exterior protocols (BGP, EGP), the autonomous system advertised to the peer is specified by the global **autonomoussystem** clause unless overridden by the **asout** parameter. The incoming autonomous system number is not verified unless **asin** is specified. Specifying **metricout** fixes the outgoing metric for all routes propagated to this peer. If the peer does not share a network, **interface** can be used to specify which interface address to use when communicating with this peer and **gateway** can be used to specify the next hop to use for all routes learned from this peer. An internal default is generated when routing information is learned from a peer unless the **nogendefault** parameter is specified.

Routing Information Protocol (RIP):

```
rip yes|no|on|off|quiet|pointopoint|supplier[ {
    preference preference ;
    defaultmetric metric ;
    interface interface_list[noripin] [noripout] ;
        ...
    trustedgateways gateway_list ;
    sourcegateways gateway_list ;
} ] ;
```

If **yes** or **on** is specified, RIP assumes **quiet** if there is only one interface and **supplier** if there are more than two or more. **quiet** specifies that no **RIP** packets are to be generated. **supplier** specifies that RIP packets are to be generated. **pointopoint** specifies that RIP packets are to be sent only to gateways listed in the **sourcegateways** clause. If the RIP clause is not specified, the default is **on**.

The default *metric* is 16, the default *preference* is 100.

HELLO Protocol:

```
hello yes|no|on|off|quiet|pointopoint|supplier [ {
preference preference ;
defaultmetric metric ;
interface interface_list [nohelloin] [nohelloout] ;
    ...
trustedgateways gateway_list ;
sourcegateways gateway_list ;
} ] ;
```

If **yes** or **on** is specified, HELLO assumes **quiet** if there is only one interface and **supplier** if there are two or more. **quiet** specifies that no HELLO packets are to be generated. **supplier** specifies that HELLO packets are to be generated. **Pointopoint** specifies that HELLO packets are to be sent only to gateways listed in the **sourcegateways** clause. If the HELLO clause is not specified the default is **off**.

The default *metric* is 30000, the default *preference* is 90.

Exterior Gateway Protocol (EGP):

```
egp yes|no|on|off [ {
    preference preference ;
    defaultmetric metric ;
    packetsize maxpacketsize ;
    group       [asin autonomous_system ]
                [asout autonomous_system ]
                [maxup number ]
                [preference preference ] {
        neighbor host
```

```
                    [metricout metric ]
                    [nogendefault]
                    [acceptdefault]
                    [propagatedefault]
                    [gateway gateway ]
                    [interface interface ]
                    [sourcenet network ]
                    [minhello min_hello ]
                    [minpoll min_poll ]
                    ;
                ...
            } ;
            ...
        } ] ;
```

packetsize specifies the size, in bytes, of the largest EGP packet to be accepted or sent. A **group** lists a group of EGP peers in one autonomous system. **maxup** specifies the maximum number of peers to be maintained in the **Up** state. **acceptdefault** and **propagatedefault** tell **gated** to accept or propagate the default network (**0.0.0.0**) in updates exchanged with an EGP neighbor. If not specified, the default network is ignored when exchanging EGP updates. **sourcenet** specifies the network to query in EGP Poll packets, this is normally the shared network. The minimum acceptable EGP hello and poll intervals can be specified with the **minhello** and **minpoll** arguments, respectively. These are both specified as a time in *seconds*, *minutes*:*seconds*, or *hours*:*minutes*:*seconds*. Any number of **group** clauses can be specified containing any number of **neighbor** clauses. Any parameters from the **neighbor** clause can be specified in the **group** clause to provide defaults for the group.

The default *metric* is 255, the default *preference* is 200.

Border Gateway Protocol (BGP):

Note that although BGP is available with this version of **gated**, *it is currently not supported by HP.*

```
bgp yes|no|on| off [ {
    preference preference ;
    defaultmetric metric ;
    peer host
        [linktype [up|down|horizontal|internal]]
        [metricout metric ]
        [asin autonomous_system ]
        [asout autonomous_system ]
        [nogendefault]
        [gateway gateway ]
        [interface interface ]
        ;
          ...
} ] ;
```

peer specifies the address of each BGP peer. A **linktype internal** is assumed if the neighbor's autonomous system is the same as my autonomous system. Otherwise **linktype horizontal** is assumed.

The default *metric* is 65 535 and the default *preference* is 150 for external BGP and 250 for internal BGP.

Redirect (ICMP):

```
redirect yes|no|on|off [ {
    preference preference ;
    interface interface_list [noicmpin] ;
    trustedgateways gateway_list ;
} ] ;
```

Controls whether gated makes routing table changes based on ICMP redirects when not functioning as a router. When functioning as a router (i.e. any interior routing protocols (RIP, HELLO) are participating in routing on any interface, ICMP redirects are disabled. When ICMP redirects are disabled,

gated must actively remove the effects of redirects from the kernel as the kernel always processes ICMP redirects.

The default preference is 20.

Static Statements

Static routes are specified with **static** clauses.

```
static {
    destination gateway gateway [preference
        preference] ;
                ...
        destination interface interface [preference
        preference] ;
                ...
} ;
```

Any number of **static** statements can be specified, each containing any number of static route definitions. The first form defines a static route through a gateway. The second defines a static interface route which is used for primitive support of multiple networks on a single interface.

The preference for static routes defaults to 50.

Control Statements

Acceptance of routes from routing protocol peers and propagation of routes to routing protocol peers are controlled by **accept** and **propagate** clauses.

Accept Clauses:

```
accept proto bgp|egp as autonomous_system [preference preference ] {
    acceptance_list
} ;
accept proto rip|hello|redirect {
    acceptance_list
} ;
accept proto rip|hello|redirect interface interface_list
    [preference preference ] {
    acceptance_list
} ;
accept proto rip|hello|redirect gateway gateway_list
    [preference preference ] {
    acceptance_list
} ;
```

where *acceptance_list* is defined as:

```
listen dest_mask [preference preference ] ;
nolisten dest_mask ;
```

If no acceptance list is specified, all routes are accepted. If one or more acceptance lists are specified, the relevant acceptance lists are scanned for a match from most-specific to least-specific (gateway, interface, protocol). If no match is found, the route is discarded. In other words, a **nolisten all** entry is assumed after all relevant acceptance lists are processed.

Propagate Clauses:

```
propagate proto bgp|egp as autonomous_system [metric metric ] {
    propagation_list
} ;
propagate proto rip|hello [metric metric ] {
    propagation_list
} ;
```

```
propagate proto rip|hello interface interface_list
    [metric metric ] {
    propagation_list
} ;
propagate proto rip|hello gateway gateway_list
    [metric metric ] {
    propagation_list
} ;
```

where *propagation_list* specifies propagation based on the origination of a destination:

```
proto bgp | egp as autonomous_system [metric metric ][ {
    announce_list
} ] ;
proto rip|hello|direct|static|default [metric metric ][ {
    announce_list
} ] ;
proto rip|hello|direct|static|default interface interface_list
    [metric metric][ {
        announce_list
    } ] ;
proto rip|hello gateway gateway_list [metric metric ][ {
    announce_list
} ] ;
```

where *announce_list* is defined as:

```
announce dest_mask [metric metric ] ;
noannounce dest_mask ;
```

If no *announce_list* is specified, all destinations are announced. If an announce list relevant to this protocol, interface, gateway, or autonomous system is specified, a `noannounce all` is assumed if no match is found after all relevant lists are examined. Therefore, an empty announce list is the equivalent of `noannounce all`. Announce lists are scanned from the most specific to the least specific, in the order specified in the config file. All lists specifying gateways are first, followed by interface lists, and finally lists that specify only the protocol.

Note that to announce routes that specify a next hop of the loopback interface (i.e., static and internally generated default routes) via RIP or HELLO it is necessary to specify the metric at some level in the propagate clause. Just setting a default metric for RIP or HELLO is not sufficient.

EXAMPLES

Several sample `gated` configuration files are provided in directory `/etc/newconfig/gated/conf`.

FILES

`/etc/gated.conf`

AUTHORS

`gated` was developed by Mark Fedor, PSI
Jeffrey C Honig, Cornell University.

SEE ALSO

arp(1m), gated(1m), ifconfig(1m), netstat(1m).

RFC 891	DCN Local-Network Protocols (HELLO)
RFC 904	Exterior Gateway Protocol Formal Specification
RFC 911	EGP Gateway under Berkeley UNIX 4.2
RFC 1058	Routing Information Protocol
RFC 1163	A Border Gateway Protocol (BGP)
RFC 1164	Application of the Border Gateway Protocol in the Internet

NAME

gettydefs - speed and terminal settings used by getty

DESCRIPTION

The `/etc/gettydefs` file contains information used by `getty` to set up the speed and terminal settings for a line (see *getty*(1M)). It supplies information on what the `login` prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by typing a **Break** character.

Each entry in `/etc/gettydefs` has the following format:

label# initial-flags # final-flags # login-prompt #next-label

Each entry is followed by a blank line. The various fields can contain quoted characters of the form `\b`, `\n`, `\c`, etc., as well as *\nnn*, where *nnn* is the octal value of the desired character. The various fields are:

label This is the string against which `getty` tries to match its second argument. It is often the speed, such as `1200`, at which the terminal is supposed to run, but it need not be (see below).

initial-flags These flags are the initial `ioctl()` settings to which the terminal is to be set if a terminal type is not specified to `getty` (see *ioctl*(2)). The flags that `getty` understands are the same as the ones listed in `/usr/include/sys/termio.h` (see *termio*(7)). Normally only the speed flag is required in the *initial-flags*. `getty` automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-flag* settings remain in effect until `getty` executes `login`.

final-flags These flags take the same values as the *initial-flags* and are set just before `getty` executes `login`. The speed flag is again required. The composite flag `SANE` takes care of most of the other flags that need to be set so that the processor and terminal are communicating in a rational fashion. The other two commonly specified *final-flags* are `TAB3`, so that tabs are sent to the terminal as spaces, and `HUPCL`, so that the line is hung up on the final close.

login-prompt This entire field is printed as the *login-prompt*. Unlike the above fields where white space is ignored (a space, tab or new-line), they are included in the *login-prompt* field.

next-label If this entry does not specify the desired speed, indicated by the user typing a **Break** character, `getty` searches for the entry with *next-label* as its *label* field and set up the terminal for those settings. Usually, a series of speeds are linked together in this fashion, into a closed set. For example, `2400` linked to `1200`, which in turn is linked to `300`, which finally is linked to `2400`.

If `getty` is called without a second argument, the first entry of `/etc/gettydefs` is used, thus making the first entry of `/etc/gettydefs` the default entry. It is also used if `getty` cannot find the specified *label*. If `/etc/gettydefs` itself is missing, there is one entry built into the command which brings up a terminal at `300` baud.

It is strongly recommended that after making or modifying `/etc/gettydefs`, it be run through `getty` with the check option to ensure that there are no errors.

EXAMPLES

The following two lines show an example of 300/1200 baud toggle, which is useful for dial-up ports:

```
1200# B1200 HUPCL # B1200 SANE IXANY IXANY TAB3 #login: #300
300# B300 HUPCL # B300 SANE IXANY IXANY TAB3 #login: #1200
```

The following line shows a typical 9600 baud entry for a hard-wired connection:

```
9600# B9600 # B9600 SANE IXANY IXANY ECHOE TAB3 #login: #9600
```

FILES

`/etc/gettydefs`

SEE ALSO

getty(1M), login(1), ioctl(2), termio(7).

NAME

glb_obj.txt - file specifying the object UUID of the Global Location Broker

SYNOPSIS

`/etc/ncs/glb_obj.txt` (Apollo Domain/OS workstations at SR10.2 or later)
`/etc/ncs/glb_obj.txt` (HP-UX systems and other UNIX systems)
`ncs$exe:glb_obj.txt` (VMS systems)
`0sb_obj.txt` (MS-DOS systems)

DESCRIPTION

The Global Location Broker (GLB) is an object identified by a Universal Unique Identifier (UUID). This UUID has a default value. The `glb_obj.txt` file allows you to override the default value by specifying a different GLB object UUID for a particular host.

The `glb_obj.txt` file is used only in special configurations that require several disjoint GLB databases (each of which is possibly replicated). In most networks and internets, there is only one GLB database (possibly replicated), and hosts do not need to have a `glb_obj.txt` file.

If a host has a `glb_obj.txt` file, the UUID in the file identifies the GLB object to which that host will direct lookups and updates. If the host runs a GLB daemon (`glbd` or `nrglbd`), the UUID also identifies the GLB object managed by that daemon, and the daemon will accept lookups and updates only for that object. By specifying different GLB object UUIDs on different sets of hosts, you can partition a network or internet into Location Broker "cells".

Location Broker cells have independent GLB databases. Each cell can be serviced by one `glbd`, one `nrglbd`, or a set of `glbd` replicas. All hosts in a cell use the same GLB object UUID. Cells need not correspond in any way to physical or logical network topology.

Managing NCS Software describes how to set up a configuration that uses Location Broker cells.

A `glb_obj.txt` file consists of one line containing the textual representation of a UUID, such as is generated by the `uuid_gen` utility. The contents of `glb_obj.txt` are identical on all hosts in a cell.

If a host does not have a `glb_obj.txt` file, it will use the default value for the GLB object UUID.

EXAMPLES

A typical `glb_obj.txt` file resembles the following:

```
437f28e72000.0d.00.00.fb.40.00.00.00
```

SEE ALSO

glbd(1M), uuid_gen(1M).

NAME

glb_site.txt - file listing possible Global Location Broker sites

SYNOPSIS

`/etc/ncs/glb_site.txt` (Apollo Domain/OS workstations at SR10.2 or later)
`/etc/ncs/glb_site.txt` (HP-UX systems and other UNIX-like systems)
`ncs$exe:glb_site.txt` (VMS systems)
`\ncs\glb_site.txt` (MS-DOS systems)

DESCRIPTION

The `glb_site.txt` file lists the network addresses of hosts where a Global Location Broker (GLB) daemon may be running.

There are two versions of the GLB daemon: `glbd` and `nrglbd`. The replicatable version, `glbd`, is provided only for Domain/OS, HP-UX, SunOS, and ULTRIX systems. For other systems, the non-replicatable version, `nrglbd`, is used. The two versions of the daemon should not coexist on a network. (For HP-UX systems, which may have both `glbd` and `nrglbd`, using only `glbd` is strongly recommended.)

Ordinarily, programs contact a GLB by broadcasting on the local network. However, some systems do not support broadcasting. Also, in certain internet configurations, not every network can have a GLB. (This typically occurs in internets that use `nrglbd`, but it can also occur in an internet that uses `glbd` if not all networks include a host that can run a `glbd`.) For hosts that cannot locate a GLB via broadcast, the `glb_site.txt` file provides a list of addresses where the host can try to directly contact a GLB.

Each line in `glb_site.txt` contains a network address where a GLB may be running. Hosts that have a `glb_site.txt` file try these addresses in order. Each address has the following form:

family : *host*

The *family* is the textual name of an address family. Possible values include `ip` and `dds`.

host is a host name. A leading `#` can be used to indicate that the host name is in the standard numeric form (such as `#192.9.8.7` or `#515c.111g`).

Blank lines and lines beginning with `#` are ignored.

If a host has a `glb_site.txt` file but does not find a GLB at any of the addresses listed in the file, the host then tries to locate one via broadcast.

See *Managing NCS Software* for information about Location Broker configuration.

EXAMPLE

The following are sample `glb_site.txt` files for the IP and DDS address families:

```
ip:piglet
ip:#192.9.8.7

dds://owl
dds:#135f.132a
```

SEE ALSO

glbd(1M).

Managing NCS Software.

NAME

group, logingroup - group file, grp.h

DESCRIPTION

`group` contains for each group the following information:

- group name
- encrypted password
- numerical group ID
- comma-separated list of all users allowed in the group

This is an ASCII file. Fields are separated by colons, and each group is separated from the next by a newline. No spaces should separate the fields or parts of fields on any line. If the password field is null, no password is associated with the group.

There are two files of this form in the system, `/etc/group` and `/etc/logingroup`. The file `/etc/group` exists to supply names for each group, and to support changing groups by means of the `newgrp` utility (see *newgrp*(1)). `/etc/logingroup` provides a default group access list for each user via `login` and `initgroups()` (see *login*(1) and *initgroups*(3C)).

The real and effective group ID set up by `login` for each user is defined in `/etc/passwd` (see *passwd*(4)). If `/etc/logingroup` is empty or non-existent, the default group access list is empty. If `/etc/logingroup` and `/etc/group` are links to the same file, the default access list includes the entire set of groups associated with the user. The group name and password fields in `/etc/logingroup` are never used; they are included only to give the two files a uniform format, allowing them to be linked together.

All group IDs used in `/etc/logingroup` or `/etc/passwd` should be defined in `/etc/group`. No user should be associated with more than `NGROUPS` (see *setgroups*(2)) groups in `/etc/logingroup`.

These files reside in directory `/etc`. Because of the encrypted passwords, these files can and do have general read permission and can be used, for example, to map numerical group IDs to names.

The group structure is defined in <`grp.h`> and includes the following members:

```
char  *gr_name;   /* the name of the group */
char  *gr_passwd; /* the encrypted group password */
gid_t gr_gid;     /* the numerical group ID */
char  **gr_mem;   /* null-terminated array of pointers
                           to member names */
```

NETWORKING FEATURES

NFS

The `/etc/group` file can contain a line beginning with a plus (+), which means to incorporate entries from Network Information Services (NIS). There are two styles of + entries: + means to insert the entire contents of NIS group file at that point, and *+name* means to insert the entry (if any) for *name* from NIS at that point. If a + entry has a non-null password or group member field, the contents of that field overide what is contained in NIS. The numerical group ID field cannot be overridden.

A group file can also have a line beginning with a minus (-), these entries are used to disallow group entries. There is only one style of - entry; an entry that consists of *-name* means to disallow any subsequent entry (if any) for *name*. These entries are disallowed regardless of whether the subsequent entry comes from the NIS or the local group file.

WARNINGS

Group ID (*gid*) 9 is reserved for the Pascal Language operating system and the BASIC Language operating system. These are operating systems for Series 300/400 computers that can co-exist with HP-UX on the same disk. Using this gid for other purposes can inhibit file transfer and sharing.

The length of each line in `/etc/group` is limited to `LINE_MAX`, as defined in <`limits.h`>. The maximum number of users per group is (`LINE_MAX` – 50)/9.

If `/etc/group` is linked to `/etc/logingroup`, group membership for a user is managed by NIS, and no NIS server is able to respond, that user cannot log in until a server does respond.

DEPENDENCIES

NFS

EXAMPLES

Here is a sample `/etc/group` file:

```
other:*:1:root,daemon,uucp,who,date,sync
-oldproj
bin:*:2:root,bin,daemon,lp
+myproject:::bill,steve
+:
```

Group `other` has a gid of 1 and members `root`, `daemon`, `uucp`, `who`, `date`, and `sync`. The group `oldproj` is ignored since it appears after the entry `-oldproj`. Also, the group `myproject` has members `bill` and `steve`, and the password and group ID of the NIS entry for the group `myproject`. All groups listed in the NIS are pulled in and placed after the entry for `myproject`.

WARNINGS

The plus (+) and minus (-) features are part of NFS. Therefore if NFS is not installed, these features cannot work.

FILES

`/etc/group`
`/etc/logingroup`

SEE ALSO

groups(1), newgrp(1), passwd(1), setgroups(2), crypt(3C), getgrent(3C), initgroups(3C), passwd(4).

WARNINGS

There is no single tool available to completely ensure that `/etc/passwd`, `/etc/group`, and `/etc/logingroup` are compatible. However, `pwck` and `grpck` can be used to simplify the task (see *pwck*(1M) and *grpck*(1M)).

There is no tool for setting group passwords in `/etc/group`.

STANDARDS CONFORMANCE

`group`: SVID2, XPG2

NAME

hosts - host name data base

DESCRIPTION

The file `/etc/hosts` associates Internet addresses with official host names and aliases. This allows a user to refer to a host by a symbolic name instead of an Internet address.

Note: This file *must* contain all addresses for local interfaces that `ifconfig` needs at boot time (see *ifconfig*(1m)), and, in HP clusters, the address of each node in the cluster. When using the name server (see *named*(1m)), or Network Information Service (see *ypserv*(1m)), this file serves only as a backup when the server is not running. In such circumstances, it is a common practice for `/etc/hosts` to contain a few addresses of machines on the local network.

`/etc/hosts` should contain a single line for each host with the following information:

<internet address> <official host name> <aliases>

If running Network Services as well as ARPA Services, an official host name consists of the first field (the node name field) of the three-field host name supported by NS. Aliases are other names by which a host is known. They can substitute for the official host name in most commands. For example:

```
192.45.36.5      hpdxsg  testhost
```

In this example, users can use remote login on hpdxsg by using the command:

```
rlogin testhost
```

instead of

```
rlogin hpdxsg
```

If your system is in a domain naming environment, an official host name consists of the full domain extended host name. For example:

```
192.45.36.5      hpdxsg.xsg.hp.com  hpdxsg  testhost
```

A line cannot start with a blank (space or tab character). Items are separated by any number or combination of space or tab characters (blanks). A # character indicates the beginning of a comment. Characters from the # to the end of the line are not interpreted by routines that search the file. Trailing blanks are allowed at the end of a line.

For the DARPA Internet network, this file is normally created from the official host database maintained at the Network Information Control Center (NIC), although local changes may be required to bring it up to date with respect to unofficial aliases and/or unknown hosts.

Network addresses are specified in the conventional Internet dot notation using the `inet_addr()` routine from the Internet address manipulation library (see *inet*(3N)). Host names can contain any printable character other than a white space, newline, or comment character.

EXAMPLES

See `/etc/hosts`.

AUTHOR

`hosts` was developed by the University of California, Berkeley.

SEE ALSO

gethostent(3N), inet(3N).

NAME

hosts.equiv, .rhosts - security files authorizing remote hosts and users on local host

DESCRIPTION

`/etc/hosts.equiv` and files named `.rhosts` in users' home directories specify remote hosts and users that are "equivalent" to the local host or user. Users from equivalent remote hosts are permitted to access a local account using `remsh` or `rcp` or to `rlogin` to the local account without supplying a password (see *remsh*(1), *rcp*(1), and *rlogin*(1)). The security defined in `hosts.equiv` is implemented by the library routine `ruserok()` (see *ruserok*(3N)). In the following, *hosts.equiv* means either `/etc/hosts.equiv` or a file `.rhosts` in a local user's home directory. Note that `.rhosts` must be owned either by the user in whose home directory it is found, or by the super-user, and must not be a symbolic link. `/etc/hosts.equiv` defines system-wide equivalency, whereas a user's `.rhosts` defines equivalency between that user and remote users to whom that user chooses to allow or deny access.

Each line of *hosts.equiv* can consist of:

- A blank line.
- A comment, beginning with a #.
- A host name, consisting of a string of any printable characters other than white space, new-line, or #.
- A host name, followed by white space, followed by a user name.

In order for a user to be granted access, both the remote host name and user name must "match" an entry in *hosts.equiv*. `/etc/hosts.equiv` is searched first. If a match is found, access is permitted. If not, it searches the `.rhosts` file if it exists in the local user's home directory. If the local user is the super-user, `/etc/hosts.equiv` is ignored.

A host name or user name matches the corresponding field in an entry in *hosts.equiv* in one of the following ways:

Literal match:
: A host name in *hosts.equiv* can literally match the official host name (not an alias) of the remote host. A user name in *hosts.equiv* can literally match the remote user name. If there is no user name in the *hosts.equiv* entry, the remote user name must literally match the local user name.

Domain-extended match:
: The remote host name to be compared with entries in *hosts.equiv* is typically the official host name returned by `gethostbyaddr()` (see *gethostbyaddr*(3N)). In a domain naming environment, this is a domain-qualified name. If a host name in *hosts.equiv* does not literally match the remote host name, the host name in *hosts.equiv* with the local domain name appended may match the remote host name.

`%`:
: any active node in an HP cluster, including the cluster server, matches the host name `%` in `hosts.equiv`. It is assumed that the names of cnodes are all in the same domain.

-name:
: If the host name in *hosts.equiv* is of this form, and if *name* literally matches the remote host name or if *name* with the local domain name appended matches the remote host name, access is denied regardless of the user name. If the user name in `hosts.equiv` is of this form, and *name* literally matches the remote user name, access is denied. Even if access is denied in this way by `/etc/hosts.equiv`, access can still be allowed by `.rhosts`.

`+`:
: Any remote host name matches the host name `+` in `hosts.equiv`; any remote user matches the user name `+`.

+@netgroup_name
: where *netgroup_name* is the name of a network group as defined in *netgroup*(4): If the host name in *hosts.equiv* is of this form, the remote host name (only) must match the specified network group according to the rules defined in *netgroup*(4) in order for the host name to match. Similarly, if the user name in *hosts.equiv* is of this form, the remote user name (only) must match the specified network group in order for the user name to match.

-@netgroup_name:
: If the host name in `hosts.equiv` is of this form, and if the remote host name (only)

matches the specified network group according to the rules defined in *netgroup*(4), access is denied. Similarly, if the user name in *hosts.equiv* is of this form, and if the remote user name (only) matches the specified network group, access is denied. Even if access is denied in this way by `/etc/hosts.equiv`, access can still be allowed by `.rhosts`.

EXAMPLES

1) `/etc/hosts.equiv` on hostA contains the line:

```
hostB
```

and `/etc/hosts.equiv` on hostB is empty. User `chm` on hostB can use `remsh` to hostA, or `rlogin` to account `chm` on hostA without being prompted for a password. `chm` will, however, be prompted for a password with `rlogin`, or denied access with `remsh`, from hostA to hostB.

If `.rhosts` in the home directory of user `chm` on hostB contains:

```
hostA
```

or

```
hostA chm
```

then user `chm` can access hostB from hostA.

2) hostA is in the domain `arg.bob.com`; hostB and hostC are in the domain `oink.bob.com`. `.rhosts` in the home directory of user `chm` on hostB contains:

```
hostC
hostA
```

User `chm` can access hostB from hostC, since `hostC.oink.bob.com` matches `hostC` with hostB's local domain `oink.bob.com` appended. But user `chm` from hostA cannot access hostB, since `hostA.arg.bob.com` does not match `hostA.oink.bob.com`. In order for user `chm` to be able to access hostB from hostA, `chm`'s `.rhosts` file on hostB must contain:

```
hostA.arg.bob.com
```

since hostA is in a different domain.

3) `.rhosts` in the home directory of user `chm` on hostA contains:

```
hostB root
```

`/etc/hosts.equiv` on hostB contains the line:

```
hostA
```

However, there is no file `.rhosts` in the home directory of user `chm` on hostB. The user `root` on hostB can `rlogin` to account `chm` on hostA without being prompted for a password, but `root` on hostA cannot `rlogin` to account `chm` on hostB.

4) `.rhosts` in the home directory of user `chm` on hostA contains:

```
+
-hostB
+ root
```

User `chm` from any host is allowed to access account `chm` on hostA. User `root` from any host except hostB can access account `chm` on hostA.

5) `/etc/hosts.equiv` on hostA contains the lines:

```
+ -chm
hostBm
```

Any user from hostB except `chm` is allowed to access an account on hostA with the same user name.

However, if `.rhosts` in the home directory of user `chm` on hostA contains:

```
hostB
```

then user `chm` from hostB can access account `chm` on hostA.

6) `/etc/hosts.equiv` on hostA contains the line:

```
+@example_group
```

The network group `example_group` consists of:

```
example_group ( , ,EXAMPLE_DOMAIN)
```

If hostA is not running Network Information Service (NFS), user `chm` on any host can access account `chm` on hostA.

If hostA is running Network Information Service (NFS), and hostA is in the domain `EXAMPLE_DOMAIN`, user `chm` on any host, whether in `EXAMPLE_DOMAIN` or not, can access account `chm` on hostA.

However, if `.rhosts` in the home directory of user `chm` on hostA contains the line:

```
-@example_group
```

and hostA is either not running Network Information Service (NFS) or is in domain `EXAMPLE_DOMAIN`, no user `chm` on any host can access the account `chm` on hostA. If hostA is running Network Information Service (NFS) but is not in the domain `EXAMPLE_DOMAIN`, this line has no effect.

7) `/etc/hosts.equiv` on hostA contains the line:

The network group example_group consists of:

```
example_group (hostB, ,)
```

All users on hostB are denied access to hostA.

However, if `.rhosts` in the home directory of a user on hostA contains any of the following lines:

```
+@example_group chm
hostB chm
+ chm
```

then user `chm` on hostB can access that account on hostA.

WARNINGS

For security purposes, the files `/etc/hosts.equiv` and `.rhosts` should exist and be readable and writable only by the owner, even if they are empty. However, if the user's home directory is mounted remotely via NFS, `.rhosts` must be readable by all. Otherwise, `remshd` and `rlogind` cannot read `.rhosts`, since they run as `root`, who normally does not have special privileges over an NFS mount (see *remshd*(1M) and *rlogind*(1M)).

The `-l` option to `remshd` and `rlogind` prevents any authentication based on `.rhosts` files for users other than the super-user.

AUTHOR

`hosts.equiv` was developed by the University of California, Berkeley.
The `%` extension was developed by HP.
The +, *-name*, *+@netgroup_name*, and *-@netgroup_name* extensions were developed by Sun Microsystems, Inc.

FILES

```
/etc/hosts.equiv
$HOME/.rhosts
```

SEE ALSO

rcp(1), remsh(1), rlogin(1), remshd(1M), rlogind(1M), rcmd(3N), netgroup(4).

NAME

inetd.conf - configuration file for inetd

DESCRIPTION

`inetd` reads its configuration information from the configuration file `/etc/inetd.conf` upon execution, and possibly at some later time in response to a `SIGHUP` signal (see *inetd*(1M)).

Each line in the file is treated either as a comment or as configuration information for a given service. Comments are denoted by a `#` at the beginning of a line. Non-comment lines contain seven or nine required fields, depending on the service name specified in the first field. Fields are separated by tabs and/or spaces. A line can be continued if it terminates with a `\`. Each configuration line in the file contains the following fields in the order indicated:

service name
socket type
protocol
`wait|nowait`
user
server program
program number (NFS RPC services only)
version number (NFS RPC services only)
server program arguments

Fields are constructed as follows:

service name `rpc` if the server is RPC-based (NFS); otherwise, the name of a valid service in file `/etc/services`. For example, `shell` for the `remsh` service (see *remsh*(1)), `login` for the `rlogin` service *rlogin*(1)), and `telnet` for the `telnet` service (see *telnet*(1)).

socket type `stream` or `dgram`, depending on whether the server socket is a stream or a datagram socket.

protocol Must be a valid protocol as given in `/etc/protocols`; for example, `tcp` or `udp`.

`wait|nowait`
Applies to datagram sockets only (other sockets should specify `nowait`).

`wait` Instructs `inetd` to execute only one datagram server for the specified socket at any one time. Datagram servers that process all datagrams on a socket and terminate by timing out are called "single-threaded". Most datagram RPC servers are single-threaded servers.

`nowait` Instructs `inetd` to execute a datagram server for a specified socket whenever a datagram arrives. Datagram servers that connect to their peers and free the socket so `inetd` can receive further datagrams are called "multi-threaded."

user User ID to be used when the server is running.

server program Absolute pathname of the program executed by `inetd` when it finds a request on the server's socket.

server program arguments
Arguments to the server program. The same as in normal use, starting with `argv`[0] which is the name of the program.

If *service name* is `rpc` (NFS RPC services), two extra fields are required. They must appear between the *server program* field and the *server program arguments* field:

program number Defines a particular service grouping and is unique.

version number Version supported by the RPC service. This number can be a single value, or a range if the program handles multiple versions; for example, `1` or `1-3`. Ranges are separated by a hyphen (`-`). Version numbers allow RPC protocols to be extended and modified, and make it possible for old and new protocols to share the same server process.

Built-in `inetd` Services

`inetd` provides several "trivial" services internally by use of built-in routines (see *inetd*(1M) for a list of these services). To configure an internal service, specify `internal` as the *server program* name, and omit the *server program arguments* field.

EXAMPLES

Configure the *shell* service to use TCP protocol, and run the server *remshd* as user `root`.

```
shell stream tcp nowait root /etc/remshd remshd
```

Configure the FTP server to timeout an inactive session after 75 seconds.

```
ftp stream tcp nowait root /etc/ftpd ftpd -t75
```

Configure an RPC-based service. Note that the *service name* field contains `rpc` and two more fields are used: the program number (100008) and version number (1).

```
rpc dgram udp wait root /usr/etc/rpc.rwalld 100008 1 rpc.rwalld
```

Configure `inetd` to use the built-in `daytime` TCP service.

```
daytime stream tcp nowait root internal
```

AUTHOR

`inetd.conf` was developed by the University of California, Berkeley.
NFS was developed by Sun Microsystems, Inc.

SEE ALSO

inetd(1M), fork(2), exec(2), inetd.sec(4), protocols(4), services(4).

NAME

inetd.sec - optional security file for inetd

DESCRIPTION

When **inetd** accepts a connection from a remote system, it checks the address of the host requesting the service against the list of hosts to be allowed or denied access to the specific service (see *inetd*(1M)). The file **inetd.sec** allows the system administrator to control which hosts (or networks in general) are allowed to use the system remotely. This file constitutes an extra layer of security in addition to the normal checks done by the services. It precedes the security of the servers; that is, a server is not started by the Internet daemon unless the host requesting the service is a valid host according to **inetd.sec**.

If file **/usr/adm/inetd.sec**, does not exist, security is limited to that implemented by the servers. **inetd.sec** and the directory **/usr/adm** should be writable only by their owners. Changes to **inetd.sec** apply to any subsequent connections.

Lines in **inetd.sec** beginning with '#' are comments. Comments are not allowed at the end of a line of data.

The lines in the file contain a service name, permission field, and the Internet addresses or official names of the hosts and networks allowed to use that service in the local host. The fields in each line are as follows:

<service name> <**allow**|**deny**> *<host or net addresses, host or net names>*

service name is the name (not alias) of a valid service in file **/etc/services**. The service name for RPC-based services (NFS) is the name (not alias) of a valid service in file **/etc/rpc**. A service name in **/etc/rpc** corresponds to a unique RPC program number.

allow|**deny** determines whether the list of remote hosts in the next field is allowed or denied access to the specified service. Multiple **allow**|**deny** lines for each service is unsupported. If there are multiple **allow**|**deny** lines for a particular service, all but the last line are ignored.

Addresses and names are separated by white space. Any mix of addresses and names is allowed. To continue a line, terminate it with \.

Host names and network names are the official names of the hosts or networks as returned by **gethostbyaddr()** or **getnetbynumber()**, respectively. Wildcard characters (*) and range characters (-) are allowed. The * and the - can be present in any of the fields of the address. An address field is a string of characters separated by a ..

EXAMPLES

Use a wildcard character to permit a whole network to communicate with the local host without having to list all the hosts in that network. For example, to allow all hosts with network addresses starting with a **10**, as well as the single host with address **192.54.24.5** to use *rlogin*:

```
login       allow   10.*  192.54.24.5
```

On a system running NFS, deny host **192.54.24.5** access to *sprayd*, an RPC-based server:

```
sprayd      deny    192.54.24.5
```

A **range** is a field containing a - character. To deny hosts in network 10 (arpa) with subnets 3 through 5 access to **remsh**:

```
shell       deny    10.3-5.*
```

The following entry denies **rlogin** access to host **cory.berkeley.edu**, any hosts on the network named **testlan**, and the host with internet address **192.54.24.5**:

```
login       deny    192.54.24.5 cory.berkeley.edu testlan
```

If a remote service is not listed in the security file, or if it is listed but it is not followed by **allow** or **deny**, all remote hosts can attempt to use it. Security is then provided by the service itself. The following lines, if present in **inetd.sec**, allow or deny access to the service indicated:

Allow all hosts to use **ftp**:

```
ftp
```

Deny all access to the **shell** service; i.e., **remsh**:

```
shell    deny
```

Allow access to the `shell` service by any host:

```
shell    allow
```

or

```
shell
```

AUTHOR

`inetd.sec` was developed by HP.

NFS was developed by Sun Microsystems, Inc.

FILES

`/usr/adm/inetd.sec`

SEE ALSO

inetd(1M), gethostent(3N), getnetent(3N), hosts(4), inetd.conf(4), networks(4), protocols(4), rpc(4), services(4).

NAME

inittab - script for the init process

DESCRIPTION

The **inittab** file supplies the script to **init**'s role as a general process dispatcher (see *init*(1M)). The process that constitutes the majority of **init**'s process dispatching activities is the line process **/etc/getty** that initiates individual terminal lines. Other processes typically dispatched by **init** are daemons and the shell.

The **inittab** file is composed of entries that are position-dependent and have the following format:

id:rstate:action:process

Each entry is delimited by a newline; however, a backslash (\) preceding a newline indicates a continuation of the entry. Up to 1024 characters per entry are permitted. Comments can be inserted in the *process* field using the *sh*(1) convention for comments. Comments for lines that spawn **getty**s are displayed by the **who** command (see *who*(1)). It is expected that they will contain some information about the line such as the location. There are no limits (other than maximum entry size) imposed on the number of entries within the **inittab** file. Entry fields are:

id A one- to four-character value used to uniquely identify an entry. Duplicate entries cause an error message to be issued, but are otherwise ignored. The use of a four-character value to identify an entry is strongly recommended (see WARNINGS below).

rstate Defines the *run*level in which this entry is to be processed. *Run levels* correspond to a configuration of processes in the system where each process spawned by **init** is assigned a *run level* or *run levels* in which it is allowed to exist. *run level*s are represented by a number ranging from **0** through **6**. For example, if the system is in *run*level **1**, only those entries having a **1** in their *rstate* field are processed.

When **init** is requested to change *run levels*, all processes that do not have an entry in the *rstate* field for the target *run level* are sent the warning signal (**SIGTERM**) and allowed a 20-second grace period before being forcibly terminated by a kill signal (**SIGKILL**). The *rstate* field can define multiple *run levels* for a process by selecting more than one *run level* in any combination from **0** through **6**. If no *run level* is specified, the process is assumed to be valid at all *run-levels*, **0** through **6**.

Three other values, **a**, **b** and **c**, can also appear in the *rstate* field, even though they are not true *run levels*. Entries having these characters in the *rstate* field are processed only when the **telinit** (see *init*(1M)) process requests them to be run (regardless of the current system *run*level). They differ from *run levels* in that **init** can never enter *run level* **a**, **b**, or **c**. Also, a request for the execution of any of these processes does not change the current *run level*.

Furthermore, a process started by an **a**, **b**, or **c** command is not killed when **init** changes levels. Processes are killed only if their line in **/etc/inittab** is marked **off** in the *action* field, their line is deleted entirely from **/etc/inittab**, or **init** goes into the *Single-User* state.

action Key words in this field tell **init** how to treat the process specified in the *process* field. Actions recognized by **init** are as follows:

respawn If the process does not exist, start the process; do not wait for its termination (continue scanning the **inittab** file). When it dies restart the process. If the process currently exists, do nothing and continue scanning the **inittab** file.

wait Upon *init*'s entering the *run level* that matches the entry's *rstate*, start the process and wait for its termination. Any subsequent reads of the **inittab** file while **init** is in the same *run level* cause **init** to ignore this entry.

once Upon **init**'s entering a *run level* that matches the entry's *rstate*, start the process; do not wait for its termination. When it dies, do not restart the process. If **init** enters a new *run level* but the process is still running from a previous *run level* change, the program is not

restarted.

`boot` Process the entry only at `init`'s boot-time read of the `inittab` file. `init` starts the process, does not wait for its termination, and when it dies, does not restart the process. In order for this instruction to be meaningful, the *rstate* should be the default or it must match `init`'s *run level* at boot time. This action is useful for an initialization function following a hardware reboot of the system.

`bootwait` Process the entry only at `init`'s boot-time read of the `inittab` file. `init` starts the process, waits for its termination and, when it dies, does not restart the process.

`powerfail` Execute the process associated with this entry only when `init` receives a power-fail signal (`SIGPWR` see *signal*(5)).

`powerwait` Execute the process associated with this entry only when `init` receives a power-fail signal (`SIGPWR`) and wait until it terminates before continuing any processing of `inittab`.

`off` If the process associated with this entry is currently running, send the warning signal (`SIGTERM`) and wait 20 seconds before forcibly terminating the process via the kill signal (`SIGKILL`). If the process is nonexistent, ignore the entry.

`ondemand` This instruction is really a synonym for the `respawn` action. It is functionally identical to `respawn` but is given a different keyword in order to divorce its association with *run levels*. This is used only with the `a`, `b`, or `c` values described in the *rstate* field.

`initdefault`
An entry with this *action* is only scanned when `init` initially invoked. `init` uses this entry, if it exists, to determine which *run level* to enter initially. It does this by taking the highest *run level* specified in the `rstate` field and using that as its initial state. If the *rstate* field is empty, this is interpreted as `0123456` causing `init` to enter *run level* `6`.

The `initdefault` entry cannot specify that `init` start in the *Single-User* state. Additionally, if `init` does not find an `initdefault` entry in `/etc/inittab`, it requests an initial *run level* from the user at reboot time.

`sysinit` Entries of this type are executed before `init` tries to access the console. It is expected that this entry will be only used to initialize devices on which `init` might attempt to obtain *run level* information. These entries are executed and waited for before continuing.

process This is a `sh` command to be executed. The entire `process` field is prefixed with `exec` and passed to a forked `sh` as `sh -c 'exec` *command*`'`. For this reason, any `sh` syntax that can legally follow `exec` can appear in the *process* field. Comments can be inserted by using `: #`*comment* syntax.

In the HP Clustered environment, `/etc/inittab` is a context-dependent file (CDF) because different cnodes have different initialization requirements. See *cdf*(4).

WARNINGS

The use of a four-character *id* is strongly recommended. Many pty servers use the last two characters of the pty name as an *id*. If an *id* chosen by a pty server collides with one used in the `/etc/inittab` file, the `/etc/utmp` file can become corrupted. A corrupt `/etc/utmp` file can cause commands such as `who` to report inaccurate information.

FILES

`/etc/inittab`

SEE ALSO
sh(1), getty(1M), exec(2), open(2), signal(5).

NAME

inode - format of an inode

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ino.h>
```

DESCRIPTION

An inode for a plain file or directory in a file system has the following structure defined by <sys/ino.h>.

```
/*  Inode structure as it appears on a disk block */

struct      dinode {
    u_short di_mode;        /* mode and type of file */
    short   di_nlink;       /* number of links to file */
    short   di_uid;         /* owner's user id */
    short   di_gid;         /* owner's group id */
    quad    di_size;        /* number of bytes in file */
    time_t  di_atime;       /* time last accessed */
    long    di_atspare;
    time_t  di_mtime;       /* time last modified */
    long    di_mtspare;
    time_t  di_ctime;       /* time of last file status change */
    long    di_ctspare;
    daddr_t di_db[NDADDR];  /* disk block addresses */
    daddr_t di_ib[NIADDR];  /* indirect blocks */
    long    di_flags;       /* status, currently unused */
    long    di_blocks;      /* blocks actually held */
    long    di_gen;         /* file generation number */
    long    di_fversion;    /* file version number */
    long    di_spare[2];    /* reserved, currently unused */
    ino_t   di_contin;      /* continuation inode number */
};
```

A continuation inode contains a file's optional access control list (ACL) entries, and has the following structure:

```
/* Continuation inode as it appears on a disk block */

struct       cinode {
    u_short ci_mode;        /* mode and type of file */
    short   ci_nlink;       /* number of links to file */
                            /* optional ACL entries */
    struct  acl_entry ci_acl[NOPTENTRIES];
    char    ci_spare[46];   /* reserved, currently unused */
};
```

For the meaning of the defined types `u_short`, `quad`, `daddr_t`, and `time_t`, see *types*(5).

Continuation inodes are distinguished from other inodes by their file type. See `/usr/include/sys/inode.h` for the definition of these values.

See `/usr/include/sys/inode.h` for the definition of inode structures for special files, pipes, or FIFOs.

WARNINGS

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

FILES

`/usr/include/sys/ino.h`

AUTHOR

AT&T, the University of California, Berkeley and HP.

SEE ALSO
stat(2), fs(4), types(5).

NAME

issue - issue identification file

DESCRIPTION

The file `/etc/issue` contains the *issue* or project identification to be printed as a login prompt. This is an ASCII file which is read by the `getty` program then written to any terminal spawned or respawned from the `inittab` file.

FILES

`/etc/issue`

SEE ALSO

getty(1), login(1).

NAME

lif - logical interchange format description

DESCRIPTION

LIF (Logical Interchange Format) is a Hewlett-Packard standard mass-storage format that can be used for interchange of files among various HP computer systems. A LIF volume contains a header (identifying it as a LIF volume) and a directory that defines the contents (i.e. files) of the volume. The size of the directory is fixed when the volume is initialized (see *lifinit*(1)) and sets an upper bound on the number of files that can be created on the volume.

HP-UX contains a set of utilities (referred to as *lif**(1)) that can be used to:

- Initialize a LIF volume (i.e. create a header and an empty directory),
- Copy files to and from LIF volumes,
- List the contents of LIF volumes,
- Remove LIF files,
- Rename LIF files.

The *lif**(1) utilities are the only utilities within HP-UX where the internal structure of a LIF volume is known. To the rest of HP-UX, a LIF volume is simply a file containing some unspecified data. The term **LIF volume** should in no way be confused with the HP-UX notion of a file system volume or mountable volume.

LIF utilities on HP-UX currently support three file types, ASCII (1), BINARY (-2) and BIN (-23951).

Three copying modes are associated with these file types:

ASCII If the copying mode is ASCII and an HP-UX file is being copied to a LIF volume, the utility strips the trailing LF (line-feed) character, and inserts two bytes of record length in front of each record. These records are then written to a LIF-formatted medium. When copying a LIF ASCII file to HP-UX the two-byte record length is stripped and a trailing LF is appended. These records are then written to the destination. In this mode of copying, the length of the file is preserved. The default file type for this mode of copying is ASCII (1).

BINARY If the copying mode is **BINARY**, and an HP-UX file is being copied to a LIF volume, the utility simply inserts two bytes for record length in front of each 1-Kbyte record. A trailing fractional block has a count reflecting the number of bytes in that block. No interpretation is placed on the content of the records. These records are then written to a LIF-format medium. When copying a LIF file to an HP-UX file in **BINARY** copying mode, the record lengths are stripped and the content of records is directly written to the destination. In this mode of copying, the length of the binary file is preserved. The default file type for this mode of copying is **BINARY** (-2).

RAW If the copying mode is RAW, and an HP-UX file is being copied to a LIF volume, the utility simply copies the raw data to the destination. File sizes that are not integer multiples of 256 bytes are padded with nulls to the next higher multiple. Therefore, *file sizes are not preserved*. When copying a LIF file to an HP-UX file in RAW mode, the information is copied directly without any interpretation placed on the content of the source. The default file type for this mode of copying is **BIN** (-23951).

A LIF volume can be created on any HP-UX file (either regular disk file or device special file) that supports random access via **lseek()** (see *lseek*(2)). *Do not mount the special file before using lif**(1) *utilities*. See *lifinit*(1) for details. Within a LIF volume, individual files are identified by 1- to 10-character file names. File names can consist of uppercase alphanumeric characters (A through Z, 0 through 9) and the underscore character (_). The first character of a LIF file name must be a letter (A through Z). The *lif**(1) utilities accept any file name (including illegal file names generated on other systems), but can only create legal names. This means that files whose names contain lowercase letters can be read but not created.

LIF file names are specified to the *lif**(1) utilities by concatenating the HP-UX path name for the LIF volume followed by the LIF file name, separating the two with a colon (**:**). For example:

/dev/fd.0:ABC specifies LIF file **ABC** accessed via HP-UX device special file **/dev/fd.0**.

myfile:ABC specifies LIF file **ABC** within HP-UX disk file **myfile**.

Note that this file-naming convention is applicable only for use as arguments to the *lif**(1) utilities, and does not constitute valid path naming for any other use within the HP-UX operating system.

*Do not mount the special file while using lif**(1) *utilities*.

SEE ALSO
lifcp(1), lifinit(1), lifls(1), lifrename(1), lifrm(1).

NAME

localedef - format and semantics of input script

DESCRIPTION

This is a description of the syntax and meaning of the script that is provided as input to the `localedef` command to create a locale (see *localedef*(1M).

The following is a list of category tags, keywords and subsequent expressions which are recognized by `localedef`. The order of keywords within a category is irrelevant with the exception of the `modifier` and `copy` keywords and other exceptions noted under the `LC_COLLATE` description. (Note that, as a convention, the category tags are composed of uppercase characters, while the keywords are composed of lowercase characters).

Category Tags and Keywords

The following keywords do not belong to any category.

`langname` String identifying the name of the language. It follows the naming convention of the `LANG` environment variable:

language [*_territory*][*.codeset*]

(see *environ*(5)). This keyword is required by `localedef` if the command line invoking `localedef` does not contain the *locale_name* (see *localedef*(1M)).

`langid` Decimal number identifying the language ID. This keyword is required by `localedef` if the command line invoking `localedef` does not contain the *locale_name* (see *localedef*(1M)). The language ID specified should be in the range of 1 to 999, and any user-defined language should assign its language ID in the range of 901 to 999.

`revision` String identifying the revision number of the `locale.inf` file. The string is restricted to contain at most 6 characters, all digits and one optional decimal point (`.`) character.

`comment_char`
Single character indicating the character to be interpreted as starting a comment line within the script. The default *comment_char* is `#`. All lines beginning with a *comment_char* are ignored.

`escape_char`
A single character indicating the character to be interpreted as an escape character within the script. The default *escape_char* is `\e` . *escape_char* is used to escape localedef metacharacters to remove special meaning and in the character constant decimal, octal, and hexadecimal formats.

The following keywords can be used in any category.

`modifier` String identifying the name of the modifier (see *environ*(5)). A *modifier* is used when a category has more than one definition. A *modifier* string is associated with each definition. Since this keyword is used to associate a modifier with a set of specifications, it must come before any keyword in that set of specifications.

`copy` A string naming another valid locale available on the system. This causes the category in the locale being created to be a copy of the same category in the named locale. Since the `copy` keyword defines the entire category, if used, it must be the only keyword in the category.

`LC_CTYPE:`

The following keywords belong to the `LC_CTYPE` category and should come between the category tag `LC_CTYPE` and `END LC_CTYPE` :

`upper` Character codes classified as uppercase letters.

`lower` Character codes classified as lowercase letters.

`digit` Character codes classified as numeric.

`space` Character codes classified as spacing (delimiter) characters.

`punct` Character codes classified as punctuation characters.

`cntrl` Character codes classified as control characters.

`blank` Character codes for printable space characters. These also must be defined in `space`.

`xdigit` Character codes classified as hexadecimal digits.

`alpha` Character codes classified as alphabetic characters. If omitted, this class is the concatenation of the `upper` and `lower` classes.

`print` Character codes classified as printable characters. If omitted, this class is the concatenation of the `upper`, `lower`, `alpha`, `digit`, `xdigit`, and `punct` classes and the <space> character.

`graph` Character codes classified as graphic characters. If omitted, this class is all characters included in `graph` except the <space> character.

`first` Character codes classified as the first bytes of two-byte characters.

`second` Character codes classified as the second bytes of two-byte characters.

`toupper` Lowercase to uppercase character relationships.

`tolower` Uppercase to lowercase character relationships.

`bytes_char` String containing the maximum number of bytes per character for the character set used for a specified language (a *langinfo*(5) item).

`alt_punct` String mapped into the ASCII equivalent string "b!"#$%&'()*+,-./:;<=>?@[\]^_'{~", where b is a blank (a *langinfo*(5) item).

`code_scheme`
Specifies the multi-byte character encoding scheme used. The operand should be a string. Currently, "HP15" and "EUC" strings are recognized. If this keyword is not specified, or the operand is a null string (""), the encoding scheme is single-byte, or HP15 if `bytes_char` is `2`. See *Native Language Support* User's Guide.

`cswidth` Defines the number of bytes contained in a character, and the number of columns per character displayed on output devices. This keyword should be specified if the encoding scheme is "EUC". EUC can be divided into 4 Supplementary Code Sets. The first SCS, Supplementary Code Set 0, contains ASCII characters and is assumed to contain 1 byte per character and require 1 column on the output devices. The operand is a string containing three ordered pairs of digits delimited by colons and commas. The format is:

X`:`*x*`,`*Y*`:`*y*`,`*Z*`:`*z*

Field	Interpretation
X	SCS 1, number of bytes
x	SCS 1, output width
Y	SCS 2, number of bytes, after SS2
y	SCS 2, output width
Z	SCS 3, number of bytes, after SS3
z	SCS 3, output width

`LC_COLLATE`:
The following keywords belong to the `LC_COLLATE` category and should come between the category tag `LC_COLLATE` and `END LC_COLLATE`. The first three keywords can be in any order, but must come before the `order_start` keyword. Any number of these three keywords can be specified.

`collating-element` <*symbol*> `from` *string*
Defines a multi-character collating element, *symbol*, composed of the characters in *string*. *String* is limited to two characters.

`collating-symbol <`*symbol*`>`
Makes *symbol* a collating symbol which can be used to define a place in the collating sequence. *Symbol* does not represent any actual character.

`order_start`
Denotes the start of the collation sequence. The directives have an effect on string collation.

The lines following the `order_start` keyword and before the `order_end` keyword contain collating element entries, one per line.

`order_end` Marks the end of the list of collating element entries.

`LC_MONETARY:`
The following keywords belong to the `LC_MONETARY` category and should come between the category tag `LC_MONETARY` and `END LC_MONETARY`. These keywords, except `crncystr`, and `mon_grouping`, are identical to the members in struct `lconv` defined in `<locale.h>` (see *localeconv*(3C)):

```
int_curr_symbol
currency_symbol
mon_decimal_point
mon_thousands_sep
positive_sign
negative_sign
int_frac_digits
frac_digits
p_cs_precedes
p_sep_by_space
n_cs_precedes
n_sep_by_space
p_sign_posn
n_sign_posn
```

`crncystr` String for specifying the currency (a *langinfo*(5) item).

`mon_grouping` A semicolon-separated list of integers. The initial integer defines the size of the group immediately preceding the decimal delimiter, and the following integers define the preceding groups (an `lconv` item).

`LC_NUMERIC:`
The following keywords belong to the `LC_NUMERIC` category and should come between the category tag `LC_NUMERIC` and `END LC_NUMERIC`:

`grouping` semicolon-separated list of integers. The initial integer defines the size of the group immediately preceding the decimal delimiter, and the following integers define the preceding groups (see struct `lconv` defined in `<locale.h>` and *localeconv*(3C)).

`decimal_point`
same as `RADIXCHAR`, a *langinfo*(5) item.

`thousands_sep`
same as `THOUSEP`, a *langinfo*(5) item.

`alt_digit` String mapped into the ASCII equivalent string "0123456789b+-.,eE", where b is a blank (a *langinfo*(5) item). The `alt_digit` keyword is a HP extension to the `localedef` POSIX standards and it has a different meaning than the `alt_digits` defined in POSIX standards.

`LC_TIME:`
The following keywords belong to the `LC_TIME` category and should come between the category tag `LC_TIME` and `END LC_TIME`. These keywords define information described in *langinfo*(5) (see *langinfo*(5)).

```
d_t_fmt
d_fmt
t_fmt
```

`t_fmt_ampm`

`day` Seven semicolon-separated strings giving names for the days of the week beginning with Sunday. Correspond to `langinfo` items `day_1` through `day_7`.

`abday` Seven semicolon-separated strings giving abbreviated names for the days of the week beginning with Sunday. Correspond to `langinfo` items `abday_1` through `abday_7`.

`mon` Twelve semicolon-separated strings giving names for the months, beginning with January. Correspond to `langinfo` items `mon_1` through `mon_12`.

`abmon` Twelve semicolon-separated strings giving abbreviated names for the months, beginning with January. Correspond to `langinfo` items `abmon_1` through `abmon_12`.

`am_pm` Two semicolon-separated strings giving the representations for `AM` and `PM`.

`year_unit`
`mon_unit`
`day_unit`
`hour_unit`
`min_unit`
`sec_unit`
`era_d_fmt`

`era` Names and dates of eras or emperors.

`LC_MESSAGES`:
The following keywords belong to the `LC_MESSAGES` category and should come between the category tag `LC_MESSAGES` and `END LC_MESSAGES`:

`yesexpr` An Extended Regular Expression matching acceptable affirmative responses to yes/no queries.

`noexpr` An Extended Regular Expression matching acceptable negative responses to yes/no queries.

`yesstr` String identifying the affirmative response for yes/no questions (a *langinfo*(5) item). This keyword is now obsolete and `yesexpr` should be used instead.

`nostr` String identifying the negative response for yes/no questions (a *langinfo*(5) item). This keyword is now obsolete and `noexpr` should be used instead.

`LC_ALL`:
The following keywords belong to the `LC_ALL` category and should come between the category tag `LC_ALL` and `END LC_ALL`:

`direction`
String indicating text direction (a *langinfo*(5) item).

`context`
String indicating character context analysis. String "null" or "0" indicates no context analysis is required. String "1" indicates Arabic context analysis required.

Keyword Operands
Keyword operands consist of character-code constants, strings, and metacharacters. The types of legal expressions are: `character lists`, `string lists`, `integer lists`, `shift`, `collating element entries`, `regular expression`, and `string`:

`character lists`
`Character list` operands follow the keywords `upper`, `lower`, `digit`, `space`, `punct`, `cntrl`, `blank`, `xdigit`, `alpha`, `print`, `graph`, `first`, and `second` and consist of single character-code constants or symbolic names separated by semicolons or a character-code range consisting of a constant or symbolic name followed by an ellipsis followed by another constant or symbolic name. The constant preceding the ellipsis must have a smaller code value than the constant following the ellipsis. A range represents a set of consecutive character codes. If the list is longer than a single line, the escape character

must be used at the end of each line as a continuation character. It is an error to use any symbolic name that is not defined in an accompanying charmap file (see *charmap*(4)).

`string lists`
`String list` operands follow the keywords `day`, `abday`, `mon`, `abmon` and `am_pm` , and consist of strings separated by semicolons. If longer than one line, the escape character must be used for continuation.

`integer lists`
`Integer list` operands follow the keywords `mon_grouping`, `int_frac_digits`, `frac_digits`, `p_cs_precedes`, `p_sep_by_space`, `n_cs_precedes`, `n_sep_by_space`, `p_sign_posn`, `n_sign_posn`, and `grouping`. An `integer list` consists of one or more decimal digits separated by semicolons.

`shift` `Shift` operands follow keywords `toupper` and `tolower`, and must consist of two character-code constants enclosed by left and right parentheses and separated by a comma. Each such character pair is separated from the next by a semicolon. For `tolower`, the first constant represents an uppercase character and the second the corresponding lowercase character. For `toupper`, the first constant represents an lowercase character and the second the corresponding uppercase character.

`collating element entry`
The `order_start` keyword is followed by collating element entries, one per line, in ascending order by collating position. The collating element entries have the form:

collation_element [*weight* [; *weight*]]

collation_element can be a character, a collating symbol enclosed in angle brackets representing a character or collating element, the special symbol `UNDEFINED` or an ellipsis (`...`).

A character stands for itself; a collating symbol can be a symbolic name for a character that is interpreted by the charmap file, a multi-character collating element defined by a `collating-element` keyword, or a collating symbol defined by the `collating-symbol keyword`.

The special symbol `UNDEFINED` specifies the collating position of any characters not explicitly defined by collating element entries. For example, if some group of characters is to be omitted from the collation sequence and just collate after all defined characters, a collating symbol might be defined before the `order_start` keyword:

```
collating-symbol <HIGH>
```

Then somewhere in the list of collating element entries:

```
UNDEFINED <HIGH>
```

Notice that there is no second weight. This means that on a second pass all characters collate by their encoded value.

An ellipsis is interpreted as a list of characters with an encoded value higher than that of the character on the preceding line and lower than that on the following line. Because it is tied to encoded value of characters, the ellipsis is inherently non-portable. If it is used, a warning is issued and no output generated unless the `-c` option was given.

The *weight* operands provide information about how the collating element is to be collated on first and subsequent passes. *Weight* can be a two-character string, the special symbol `IGNORE`, or a collating element of any of the forms specified for *collating_element* except `UNDEFINED`. If there are no *weights*, the character is collating strictly by its position in the list. If there is only one *weight* given, the character sorts by its relative position in the list on the second collation pass.

An equivalence class is defined by a series of collating element entries all having the same character or symbol in the first *weight* position. For example, in many locales all forms of the character This is represented in the collating element entries as:

```
'A'     'A';'A' # first element of equivalence class
'a'     'A';'a' # next element of class
```

Two-to-one collating elements are specified by *collating-elements* defined before the `order_start keyword`. For example, the two-to-one collating element `CH` in Spanish, would be defined before the `order_start` keyword as

```
collating element <CH> from CH
```

It would then be used in a collating element entry as `<CH>`.

A one-to-two collating element is defined by having a two-character string in one of the *weight* positions. For example, if the character 'X' collates equal to the pair "AE", the collating element entry would be:

```
'X' AE ;'X'
```

A don't-care character is defined by the special symbol `IGNORE`. For example, the dash character, '-' may be a don't care on the first collation pass. The collating element entry is:

```
'-' IGNORE;'-'
```

Symbols defined by the `collating-symbol` keyword can be used to indicate that a given character collates higher or lower than some position in the sequence. For example if all characters with an encoded value less than that of `'0'` are to collate lower than all other charactes on the first pass, and in relative order on the second pass, define a collating symbol before the `order_start` keyword:

```
collating-symbol    <LOW>
```

The first two collating element entries are then:

```
...     <LOW>;...
'0'     '0';'0'
```

This also illustrates the use of the ellipsis to indicate a range. The first ellipsis is interpreted as "all characters in the encoded character set with a value lower than '0'"; the second ellipsis means that all characters in the range defined by the first collate in relative order.

`regular expression`
: `Regular expression` operands follow the keywords `yesexpr` and `noexpr`, which can be Extended Regular Expressions as described in *regexp*(5).

`string`
: `String` operands follow all *langinfo*-type, *lconv*-type and `era` keywords except `mon_grouping` and `grouping`. Each expression is a string (see `Strings` section below).

The expressions following the *langinfo*-type keywords define the strings associated with *items* in *langinfo*(5). Each expression consists of a string to be associated with the *item* identified by the keyword.

The expressions following the *lconv*-type keywords define the strings associated with *members* of `lconv` struct in *localeconv*(3C). Each expression consists of a string to be associated with the *member* identified by the keyword.

Each expression following the keyword `era` defines how the years are counted and displayed for one era (or emperor's reign). The expressions must be in the following format:

direction:*offset*:*start_date*:*end_date*:*name*:*format*

where:

direction
: Either a + or - character. The + character indicates the time axis should be such that the years count in the positive direction when moving from the starting date towards the ending date. The - character indicates the time axis should be such that the years count in the negative direction when moving from the starting date towards the ending date.

offset A number in the range `[SHRT_MIN, SHRT_MAX]` indicating the number of the first year of the era.

start_date A date in the form *yyyy/mm/dd* where *yyyy*, *mm*, and *dd* are the year, month and day numbers, respectively, of the start of the era. Years prior to the year 0 A.D. are represented as negative numbers. For example, an era beginning March 5th in the year 100 B.C. would be represented as `-100/3/5`. Years in the range `[SHRT_MIN+1, SHRT_MAX-1]` are supported.

end_date The ending date of the era in the same form as the *start_date* above or one of the two special values `-*` or `+*`. A value of `-*` indicates the ending date of the era extends to the beginning of time while `+*` indicates it extends to the end of time. The ending date can be chronologically either before or after the starting date of an era. For example, the expressions for the Christian eras A.D. and B.C. would be:

```
+:0:0000/01/01:+*:A.D.:%o %N
+:1:-0001/12/31:-*:B.C.:%o %N
```

name
A string representing the name of the era which is substituted for the `%N` directive of `date` and `strftime()` (see *date*(1) and *strftime*(3C)).

format
A string for formatting the `%E` directive of *date*(1) and *strftime*(3C). This string is usually a function of the `%o` and `%N` directives. If *format* is not specified, the string specified for the `LC_TIME` category keyword `era_d_fmt` (see above) is used as a default.

Constants

Constants represent character codes in the operands. They can used in the following forms:

decimal constants
An escape character followed by a `'d'` followed by up to three decimal digits.

octal constants An escape character followed by up to three octal digits.

hexadecimal constants
An escape character followed by a `'x'` followed by two hexadecimal digits.

character constants
A single character enclosed in single quotes or separated from any other single character by a semicolon, comma or <blank> having the numerical value of the character in the machine's character set.

symbolic names
A string enclosed between < and > is a symbolic name. `localedef` scripts can be written entirely in symbolic names and have them interpreted according to a charmap file. This aids portability of `localedef` scripts between different encoded character sets (see *charmap*(4)).

Symbolic names can be defined within a script by the `collating-element` and `collating-symbol` keywords. These are not character constants. It is an error if such an internally defined symbolic name collides with one defined in a charmap file.

Strings

Strings are used in `string` and `string list` operands. A string is a sequence of zero or more characters either surrounded by double quotes (") or delimited by semicolons or <blank>s Within a string, the double-quote character must be preceded by an escape character. The following escape sequences also can be used:

\n newline

\t horizontal tab

\b backspace

\r	carriage return
\f	form feed
\\	backslash
\'	single quote
\ddd	bit pattern

The escape \ddd consists of the escape character followed by 1, 2, or 3 octal digits specifying the value of the desired character. Also, an escape character (\) and an immediately-following newline are ignored.

Although the backslash (\) has been used for illustration, another escape character can be substituted by the `escape_char` keyword.

Metacharacters

Metacharacters are characters having a special meaning to *localedef* in operands. To escape the special meaning of these characters, surround them with single quotes or precede them by an escape character. *localedef* meta-characters include:

<	Indicates the beginning of a symbolic name.
>	Indicates the end of a symbolic name.
(	Indicates the beginning of a character shift pair following the `toupper` and `tolower` keywords.
)	Indicates the end of a character shift pair.
,	Used to separate the characters of a character shift pair.
"	Used to quote strings.
;	Used as a separator in list operands.

`escape character`
: Used to escape special meaning from other metacharacters and itself. It is backslash (\) by default, but can be redefined by the `escape_char` keyword.

Comments

Comments are lines beginning with a comment character. The comment character is pound sign (#) by default, but can be redefined by the `comment_char` keyword. Comments and blank lines are ignored.

Separators

Separator characters include blanks and tabs. Any number of separators can be used to delimit the keywords, metacharacters, constants and strings that comprise a *localedef* script except that all characters between < and > are considered to be part of the symbolic name even they are <blank>s.

GRAMMAR

The following is a yacc-style grammar for a localedef script as specified by POSIX.2. It omits some elements that are used in *localedef* in HP-UX but are not required by POSIX.2 such as langname, langid, etc.

The following tokens are processed (in addition to those string constants shown in the grammar):

`LOC_NAME`	String of characters representing the name of a locale.
CHAR	Any single character.
`NUMBER`	Decimal number represented by one or more decimal digits.
`COLLSYMBOL`	String of characters in the set of visible glyphs defined in table 2-3, enclosed between angle brackets. The string must not duplicate any charmap symbol defined in the current charmap (if it exists).
`CHARSYMBOL`	Symbolic name, enclosed between angle brackets, from the current charmap (if it exists).
`OCTAL_CHAR`	One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an `escape_char` (normally a backslash) followed by two or three octal digits.

HEX_CHAR	One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an **escape_char** followed by the constant **x** and two hexadecimal digits.
DECIMAL_CHAR	One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an **escape_char** followed by a **d** and two, three or four decimal digits.
ELLIPSIS	The string "...".
EXTENDED_REG_EXP	An extended regular expression (see *regexp*(5).
EOL	The line termination character (new-line character).

This subclause presents the grammar for the locale definition.

```
%token  LOC_NAME
%token  CHAR
%token  NUMBER
%token  COLLSYMBOL COLLELEMENT
%token  CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
%token  ELLIPSIS
%token  EXTENDED_REG_EXP
%token  EOL

%start  locale_definition

%%

locale_definition       : global_statements locale_categories
        |               locale_categories
        ;
global_statements       :  global_statements symbol_redefine
        | symbol_redefine
        ;
symbol_redefine : 'escape_char'  CHAR EOL
        | 'comment_char' CHAR EOL
        ;
locale_categories       : locale_categories locale_category
        | locale_category
        ;
locale_category : lc_ctype
        | lc_collate
        | lc_messages
        | lc_monetary
        | lc_numeric
        | lc_time
        ;

/*      The following grammar rules are common to all categories */

char_list       : char_list char_symbol
        | char_symbol
        ;
char_symbol     : CHAR
        | CHARSYMBOL
        | OCTAL_CHAR
        | HEX_CHAR
        | DECIMAL_CHAR
        ;
locale_name     : LOC_NAME
        | '"' LOC_NAME '"'
        ;
```

```
/*      The following is the LC_CTYPE category grammar */
lc_ctype        : ctype_hdr ctype_keywords      ctype_tlr
        | ctype_hdr 'copy' locale_name EOL ctype_tlr
        ;
ctype_hdr       : 'LC_CTYPE' EOL
        ;
ctype_body      : 'copy' locale_name EOL
        | ctype_keywords
        ;
ctype_keywords  : ctype_keywords ctype_keyword
        | ctype_keyword
        ;
ctype_keyword   : charclass_keyword charclass_list EOL
        | charconv_keyword charconv_list EOL
        ;
charclass_keyword       : 'upper'
        | 'lower'
        | 'alpha'
        | 'digit'
        | 'alnum'
        | 'xdigit'
        | 'space'
        | 'print'
        | 'graph'
        | 'blank'
        | 'cntrl'
        ;
charclass_list  : charclass_list ';' char_symbol
        | charclass_list ';' ELLIPSIS ';' char_symbol
        | char_symbol
        ;
charconv_keyword        : 'toupper'
        | 'tolower'
        ;
charconv_list   : charconv_list ';' charconv_entry
        | charconv_entry
        ;
charconv_entry  : '(' char_symbol ',' char_symbol ')'
        ;
ctype_tlr       : 'END' 'LC_CTYPE' EOL
        ;
/*      The following is the LC_COLLATE category grammar */
lc_collate      : collate_hdr collate_keywords collate_tlr
        | collate_hdr 'copy' locale_name EOL collate_tlr
        ;
collate_hdr     : 'LC_COLLATE' EOL
        ;
collate_keywords:               order_statements
        | opt_statements order_statements
        ;
opt_statements  : opt_statements collating_symbols
        | opt_statements collating_elements
        | collating_symbols
        | collating_elements
        ;
collating_symbols       : 'collating-symbol' COLLSYMBOL EOL
        ;
collating_elements      : 'collating-element' COLLELEMENT
```

```
        'from' '"' char_list '"' EOL
        ;
order_statements        : order_start collation_order order_end
        ;
order_start     : 'order_start' EOL
        ;
collation_order : collation_order collation_entry
        | collation_entry
        ;
collation_entry : COLLSYMBOL EOL
        | collation_element weight_list EOL
        | collation_element             EOL
        ;
collation_element       : char_symbol
        | COLLELEMENT
        | ELLIPSIS
        | 'UNDEFINED'
        ;
weight_list     : weight_list ';' weight_symbol
        | weight_list ';'
        | weight_symbol
        ;
weight_symbol   : char_symbol
        | COLLSYMBOL
        | '"' char_list '"'
        | ELLIPSIS
        | 'IGNORE'
        ;
order_end       : 'order_end' EOL
        ;
collate_tlr     : 'END' 'LC_COLLATE' EOL
        ;
/*      The following is the LC_MESSAGES category grammar */
lc_messages     : messages_hdr messages_keywords        messages_tlr
        | messages_hdr 'copy' locale_name EOL messages_tlr
        ;
messages_hdr    : 'LC_MESSAGES' EOL
        ;
messages_keywords       : messages_keywords messages_keyword
        | messages_keyword
        ;
messages_keyword        : 'yesexpr'  '"' EXTENDED_REG_EXP '"' EOL
        | 'noexpr' '"' EXTENDED_REG_EXP '"' EOL
        ;
messages_tlr    : 'END' 'LC_MESSAGES' EOL
        ;
/*      The following is the LC_MONETARY category grammar */
lc_monetary     : monetary_hdr monetary_keywords        monetary_tlr
        | monetary_hdr 'copy' locale_name EOL   monetary_tlr
        ;
monetary_hdr    : 'LC_MONETARY' EOL
        ;
monetary_keywords       : monetary_keywords monetary_keyword
        | monetary_keyword
        ;
monetary_keyword        : mon_keyword_string mon_string EOL
        | mon_keyword_char NUMBER EOL
```

```
        | mon_keyword_char '-1' EOL
        | mon_keyword_grouping mon_group_list EOL
        ;
mon_keyword_string      : 'int_curr_symbol'
        | 'currency_symbol'
        | 'mon_decimal_point'
        | 'mon_thousands_sep'
        | 'positive_sign'
        | 'negative_sign'
        ;
mon_string      :'"' char_list '"'
        | '""'
        ;
mon_keyword_char        : 'int_frac_digits'
        | 'frac_digits'
        | 'p_cs_precedes'
        | 'p_sep_by_space'
        | 'n_cs_precedes'
        | 'n_sep_by_space'
        | 'p_sign_posn'
        | 'n_sign_posn'
        ;
mon_keyword_grouping    : 'mon_grouping'
        ;
mon_group_list  : NUMBER
        | mon_group_list ';' NUMBER
        ;
monetary_tlr    : 'END' 'LC_MONETARY' EOL
        ;

/*      The following is the LC_NUMERIC category grammar */

lc_numeric      : numeric_hdr numeric_keywords  numeric_tlr
        | numeric_hdr 'copy' locale_name EOL    numeric_tlr
        ;
numeric_hdr     : 'LC_NUMERIC' EOL
        ;
numeric_keywords        : numeric_keywords numeric_keyword
        | numeric_keyword
        ;
numeric_keyword : num_keyword_string num_string EOL
        | num_keyword_grouping num_group_list EOL
        ;
num_keyword_string      : 'decimal_point'
        | 'thousands_sep'
        ;
num_string      :  '"' char_list '"'
        | '""'
        ;
num_keyword_grouping    : 'num_grouping'
        ;
num_group_list  : NUMBER
        | num_group_list ';' NUMBER
        ;
numeric_tlr     : 'END' 'LC_TIME' EOL
        ;

/*      The following is the LC_TIME category grammar */

lc_time : time_hdr time_keywords        time_tlr
        | time_hdr 'copy locale_name    EOL time_tlr
```

```
        ;
time_hdr        : 'LC_TIME' EOL
        ;
time_keywords   : time_keywords time_keyword
        | time_keyword
        ;
time_keyword    : time_keyword_name time_list EOL
        | time_keyword_fmt time_string EOL
        | time_keyword_opt time_list EOL
        ;
time_keyword_name       : 'abday'
        | 'day'
        | 'abmon'
        | 'mon'
        ;
time_keyword_fmt        : 'd_t_fmt'
        | 'd_fmt'
        | 't_fmt'
        | 'am_pm'
        | 't_fmt_ampm'
        ;
time_keyword_opt        : 'era'
        | 'era_year'
        | 'era_d_fmt'
        | 'alt_digits'
        ;
time_list       : time_list ';' time_string
        | time_string
        ;
time_string     : '"' char_list '"'
        ;
time_tlr        : 'END' 'LC_TIME' EOL
        ;
```

EXAMPLES

The following `localedef` script creates the `locale.inf` file for the `american` language using the ROMAN8 code set:

```
# language: american
# code set: ROMAN8

langname        "american"
langid          1
revision        "1.1"
escape_char     \
comment_char    '#'

##################################################
# Set up the LC_CTYPE category of the table

LC_CTYPE
upper   'A'...'Z';
        \xa1...\xa7;\xad;\xae;\xb1;\xb4;\xb6; \
        \xd0;\xd2;\xd3;\xd8;\xda...\xdc; \
        \xde..\xe1;\xe3;\xe5...\xe9; \
        \xeb;\xed;\xee;\xf0
lower   'a'...'z';
        \xb2;\xb5;\xb7;\xc0...\xcf; \xd1; \
        \xd4...\xd7;\xd9; \xdd;\xde ;\xe2; \
        \xe4;\xea;\xec;\xef;\xf1
digit   '0'...'9'
space   ' ';\x9...\xd
```

```
punct   '!'...'/';':' - '@';
        '['...'`';'{' - '~' \
        \d168...\d172;\d175;\d176;\d179;
        \d184...\d191;\d242...\d254
cntrl   \x0...\x1f;\x7f;
        \200...\240;\377
blank   ' ';\t
xdigit  '0'...'9';'a'...'f';
        'A'...'F'

# isfirst and issecond are irrelevant here
# alpha, graph and print get default values

tolower ( 'A','a' );( 'B','b' );( 'C','c' ); \
        ( 'D','d' );( 'E','e' );( 'F','f' );
        ( 'G','g' );( 'H','h' ); \
        (\x49,\x69);  (\x4a,\x6a);
        (\113,\153);  (\114,\154);
        (\d77,\d109); (\d78,\d110);
        ( 'O','o' );  ( 'P','p' );
        ( 'Q','q' );( 'R','r' ); ( 'S','s' ); \
        ( 'T','t' );( 'U','u' );( 'V','v' ); \
        ( 'W','w' );( 'X','x' );( 'Y','y' ); \
        ( 'Z','z' );(\xa1,\xc8);(\xa2,\xc0); \
        (\xa3,\xc9);(\xa4,\xc1);(\xa5,\xcd); \
        (\xa6,\xd1);(\xa7,\xdd);(\xad,\xcb); \
        (\xae,\xc3);(\xb1,\xb2);(\xb4,\xb5); \
    (\xb6,\xb7);(\xd0,\xd4);(\xd2,\xd6); \
    (\xd3,\xd7);(\xd8,\xcc);(\xda,\xce); \
    (\xdb,\xcf);(\xdc,\xc5);(\xdf,\xc2); \
    (\xe0,\xc4);(\xe1,\xe2);(\xe3,\xe4); \
    (\xe5,\xd5);(\xe6,\xd9);(\xe7,\xc6); \
    (\xe8,\xca);(\xe9,\xea);(\xeb,\xec); \
    (\xed,\xc7);(\xee,\xef);(\xf0,\xf1)

# toupper is the reverse of tolower

bytes_char      "1"
alt_punct       ""
code_scheme     ""
END LC_CTYPE

##################################################
# Set up the LC_COLLATE category of the table

# dictionary collating sequence:
# spaces, decimal digits,
# alphabetic characters, punctuation,
# control characters

LC_COLLATE
modifier      "nofold"
order_start
' '         ' ';' '
\xa0        \xa0; \xa0
'0'         '0';'0'
# '1' through '8' in numerical order
'9'         '9';'9'
# Equivalence class of 'A' starts here
'A'         'A';'A'
# One-to-two, AE ligature
\xd3       'A';"AE"
\xe0       'A';\xe0
```

```
\xa1        'A';\xa1
\xa2        'A';\xa2
\xd8        'A';\xd8
\xd0        'A';\xd0
# Equivalence class of 'A' ends here
\xe1        'A';\xe1
´B´          'B';'B'
´C´          'C';'C'
\xb4        'C';\xb4
´D´          'D';'D'
\xe3        'D';\xe3
´E´          'E';'E'
\xdc        'E';\xdc
\xa3        'E';\xa3
\xa4        'E';\xa4
\xa5        'E';\xa5
´F´          'F';'F'
´G´          'G';'G'
´H´          'H';'H'
´I´          'I';'I'
\xe5        'I';\xe5
\xe6        'I';\xe6
\xa6        'I';\xa6
\xa7        'I';\xa7
´J´          'J';'J'
´K´          'K';'K'
´L´          'L';'L'
´M´          'M';'M'
´N´          'N';'N'
\xb6        'N';\xb6
´O´          'O';'O'
\xe7        'O';\xe7
\xe8        'O';\xe8
\xdf        'O';\xdf
\xda        'O';\xda
\xe9        'O';\xe9
\xd2        'O';\xd2
´P´          'P';'P'
´Q´          'Q';'Q'
´R´          'R';'R'
# Remainder of LC_COLLATE omitted for space considerations
order_end
END LC_COLLATE
##################################################
# Set up the LC_MONETARY category of the table
LC_MONETARY
int_curr_symbol      "USD "
currency_symbol      "$"
mon_decimal_point    "."
mon_thousands_sep    ","
mon_grouping         3;0
positive_sign        ""
negative_sign        "-"
int_frac_digits      "2"
frac_digits          "2"
p_cs_precedes        "1"
p_sep_by_space       "0"
```

```
n_cs_precedes          "1"
n_sep_by_space         "0"
p_sign_posn            "1"
n_sign_posn            "1"
crncystr               "-US$"
END LC_MONETARY

##################################################
# Set up the LC_NUMERIC category of the table

LC_NUMERIC
grouping         3;0
thousands_sep    ","
decimal_point    "."
alt_digit       ""
END LC_NUMERIC

##################################################
# Set up the LC_TIME category of the table

LC_TIME
# date & time format string
d_t_fmt     "%a, %b %.1d, %Y %I:%M:%S %p"
# date format string
d_fmt       "%a, %b %.1d, %Y"
# time format string
t_fmt       "%I:%M:%S"
# 12-hr time format
t_fmt_ampm "%I:%M:%S %p"

# Days of week
day      "Sunday";"Monday";"Tuesday";"Wednesday";"Thursday"; \
"Friday";"Saturday"

# Weekday abbreviations
abday    "Sun";"Mon";"Tue";"Wed";"Thu";"Fri";"Sat"

# Month names
mon      "January";"February";"March";"April";"May";"June"; \
"July";"August";"September";"October";"November";"December"

# month abbreviations
abmon  "Jan";"Feb";"Mar";"Apr";"May";"Jun";"Jul";"Aug"; \
"Sep";"Oct";"Nov";"Dec"

# AM, PM strings
am_pm  "AM";"PM"

year_unit       ""
mon_unit        ""
day_unit        ""
hour_unit       ""
min_unit        ""
sec_unit        ""

# There is no era or emperor year for the "american" language,
# but here is an example of the "japanese" era_d_fmt and era specification:

# normal era format string
era_d_fmt   "%N%onen"
era  "+:2:1990/01/01:+*:Heisei"
# special fmt for 1st year
     "+:1:1989/01/08:1989/12/31:Heisei:%Ngannen"
     "+:2:1927/01/01:1989/01/07:Shouwa"
# special fmt for 1st year
```

```
     "+:1:1926/12/25:1926/12/31:Shouwa:%Ngannen"
     "+:2:1913/01/01:1926/12/24:Taishou"
# special fmt for 1st year
     "+:1:1912/07/30:1912/12/31:Taishou:%Ngannen"
     "+:2:1869/01/01:1912/07/29:Meiji"
# special fmt for 1st year
     "+:1:1868/09/08:1868/12/31:Meiji:%Ngannen"
# revert to regular year numbering
# for years prior to the supported eras
     "-:1868:1868/09/07:-*::%o"
END LC_TIME

##################################################
# Set up the LC_MESSAGES category of the table

LC_MESSGAES
# could be "[yY][eE][sS]"
yesexpr "yes"
noexpr  "no"
END LC_MESSAGES

##################################################
# Set up the LC_ALL category of the table

LC_ALL
# left-to-right orientation
direction       ""
context         ""
END LC_ALL
```

NAME

lvmpvg - store physical volume group information for LVM

SYNOPSIS

`/etc/lvmpvg`

DESCRIPTION

`lvmpvg` is an ASCII file that stores the volume-group information for all of the physical volume groups in the system. The information is stored in a hierarchical format. First, it starts with a volume group under which multiple physical volume group can exist. Under each physical volume group, a list of physical volumes can be specified. There must be at least one physical volume group in each volume group that appears in this file. The physical-volume-group name must be unique within the corresponding volume group, although it is permissible to use a common physical volume group name across different volume groups. There can be as many volume groups in this file as there are in the system.

Instead of using the `vgcreate` and `vgextend` commands, the administrator can edit this file to create and extend physical volume groups. However, care must be taken to ensure that all physical volumes to be included in the file have already been defined in their respective volume groups by previous use of `vgcreate` or `vgextend`.

The `lvmpvg` file format is structured as follows: `VG` and `PVG` are keywords that stand for Volume Group and Physical Volume Group respectively. *No comments are allowed in this file.*

```
VG        vol_group_name
PVG       physical_vol_group_name
physical_vol_path
        .
        .
        .
PVG       physical_vol_group_name
physical_vol_path
        .
        .
        .
VG        vol_group_name
PVG       physical_vol_group_name
physical_vol_path
        .
        .
        .
```

EXAMPLES

The following example shows an *lvmpvg* file containing two volume groups: the first containing two physical volume groups, each with two physical volumes defined in it; the second containing three physical volume groups, each with one physical volume defined in it.

```
VG      /dev/vg00
PVG     PVG0
/dev/dsk/c2d0s2
/dev/dsk/c2d1s2
PVG     PVG1
/dev/dsk/c3d0s2
/dev/dsk/c3d1s2
VG      /dev/vg01
PVG     PVG0
/dev/dsk/c4d0s7
PVG     PVG1
/dev/dsk/c5d0s7
PVG     PVG2
/dev/dsk/c6d0s7
```

SEE ALSO

vgcreate(1M), vgextend(1M), vgreduce(1M), vgremove(1M).

NAME

magic - magic numbers for HP-UX implementations

SYNOPSIS

```
#include <magic.h>
```

DESCRIPTION

The `magic.h` file localizes all information about HP-UX "magic numbers" in one file, thus facilitating uniform treatment of magic numbers. This file specifies the location of the magic number in a file (always the start of the file) and the structure of the magic number:

```
struct magic_number {
    unsigned short    system_id;
    unsigned short    file_type;
};
typedef       struct    magic_number MAGIC;
```

`magic.h` includes definitions for the system IDs of all HP machines running HP-UX, and file types that are common to all implementations. There may be additional implementation-dependent file types. The predefined file types are:

```
/* for object code files */
    #define RELOC_MAGIC     0x106   /* relocatable only */
    #define EXEC_MAGIC      0x107   /* normal executable */
    #define SHARE_MAGIC     0x108   /* shared executable */
    #define DEMAND_MAGIC    0x10B   /* demand-load executable */
    #define LISP_MAGIC      0x10C   /* compiled Lisp */
    #define DL_MAGIC        0x10D   /* dynamic load library */
    #define SHL_MAGIC       0x10E   /* shared library */
    #define HPE_MAGIC       0x150   /* HPE boot image */
```

The values for `system_id` are defined in *model*(4).

WARNINGS

Files managed by `cpio` use a different form of magic number that is incompatible with <`magic.h`>.

SEE ALSO

ar(1), ld(1), a.out(4), ar(4), model(4).

NAME

master - master device information table

DESCRIPTION

This file contains lines of various forms and is used by `config` to obtain device information that enables it to generate the configuration file (see *config*(1M)).

Software drivers are defined as follows:

Field 1 — Device name, used in the user-specified `dfile` (8 characters maximum).

Field 2 — Handler name, used by the kernel to prefix routines such as `cs80_read`, `lp_write`, and others (8 characters maximum).

Field 3 — Element characteristics: Five bits make up the mask

- Bit 1: card
- Bit 2: specified only once
- Bit 3: required driver
- Bit 4: block device
- Bit 5: character device (LSB)

Field 4 — Functions for the device; 12 bits make up the mask

- Bit 1 option1 handler (Series 700 only)
- Bit 2 dump handler
- Bit 3 size handler
- Bit 4 link routine
- Bit 5 `open` handler
- Bit 6 `close` handler
- Bit 7 `read` handler
- Bit 8 `write` handler
- Bit 9 `ioctl` handler
- Bit 10 `select` handler
- Bit 11 `seltru` handler
- Bit 12 `C_ALLCLOSES` flag (LSB)

Field 5 — Major device number if a block-type device; otherwise –1.

Field 6 — Major device number if a character-type device; otherwise –1.

Aliases for names are defined as follows:

Field 1 — Alias name => product number (8 characters maximum)

Field 2 — Device name (8 characters maximum)

Parameters are defined as follows:

Field 1 — Parameter name as used in the user-specified `dfile` (20 characters maximum).

Field 2 — Parameter name as used in the `#define` statement in `conf.c` (20 characters maximum).

Field 3 — Default value for the parameter (60 characters maximum).

Field 4 — Minimum value for the parameter (60 characters maximum).

SEE ALSO

config(1M).

NAME

mirrortab - mirror disk-log format

Remarks:

Mirror facilities require installation of optional DataPair/800 software (not included in the standard HP-UX operating system) before they can be used.

DESCRIPTION

`mirrortab` is a file describing all mirrors on the system. This file is created and maintained by the `mirrorlog` daemon (see *mirrorlog*(1M).

There is one line in the file for each mirror. Each line contains the following blank-separated fields in the order shown (see *mirror*(1M) for definitions of terms):

primary device — Pathname of the primary section of the mirror. The name of the mirror is the same as the name of the primary device.

primary state — State of the primary section of the mirror. Possible values are **ONLINE**, indicating that the section is in operation; **OFFLINE**, indicating that the section is not available for use; and **REIMAGE**, indicating that the section is in the process of being brought into agreement with its **ONLINE** counterpart.

secondary device — Pathname of the secondary section of the mirror pair.

secondary state — State of the secondary section of the mirror pair. Possible values are as described for *primary state*.

fail — Mirror fail flag. Possible values are **GOOD**, indicating that neither of the sections making up the mirror pair has experienced a failure, and **FAIL**, indicating that one of the sections making up the mirror pair has failed, and thus is **OFFLINE**.

clean — Mirror-clean flag. Possible values are **CLEAN**, indicating that the two sections making up the mirror pair are guaranteed to be identical, and **DIRTY**, indicating that the sections *may* differ.

EXAMPLE

Here is a sample that resembles a typical `mirrortab` file:

```
/dev/rdsk/c2001d0s1 ONLINE  /dev/rdsk/c2002d0s1  REIMAGE GOOD DIRTY
/dev/rdsk/c2001d0s4 ONLINE  /dev/rdsk/c2002d0s4  ONLINE  GOOD CLEAN
/dev/rdsk/c2001d0s5 OFFLINE /dev/rdsk/c2002d0s5  ONLINE  FAIL DIRTY
```

SEE ALSO

mirror(1M), mirrorlog(1M), brc(1M).

NAME

mnttab - mounted file system table

SYNOPSIS

```
#include <mntent.h>
```

DESCRIPTION

mnttab resides in directory **/etc** and contains a table of devices mounted by the **mount** command (see *mount*(1M)). The file contains a line of information for each mounted filesystem which, with the exception of the **cnode_id** field, is structurally identical to the contents of **/etc/checklist** described by *checklist*(4).

There are a number of lines of the form:

special_file_name dir type opts freq passno mount_time cnode_id

consisting of entries similar to:

```
/dev/dsk/c0d0s0  /  hfs  rw  0  1  537851723  1
```

/etc/mnttab is accessed by programs that use **getmntent()** (see *getmntent*(3X)), It should never be manually edited, nor should **setmnt** ever be used to create invalid entries in **/etc/mnttab** (see *setmnt*(1M)).

mount_time contains the time the file system was mounted using **mount**. Its value is the number of seconds since the Epoch (00:00:00 Coordinated Universal Time, January 1, 1970 (see *time*(2).

mount and **umount** rewrite the **mnttab** file whenever a file system is mounted or unmounted if **mnttab** is found to be out of date with the mounted file system table maintained internally by the HP-UX kernel. **syncer** also updates **mnttab** if it is out of date (see *syncer*(1M).

WARNINGS

The table is provided only as a means for programs to return information about mounted file systems.

/etc/mnttab should never be manually edited. Any manual changes made to **/etc/mnttab** are overwritten without warning by **syncer**, **mount**, and **umount**.

AUTHOR

mnttab was developed by the University of California, Berkeley, Sun Microsystems, Inc., and HP.

FILES

/etc/mnttab

SEE ALSO

mount(1M), getmntent(3X), checklist(4).

NAME

model - HP-UX machine identification

SYNOPSIS

```
#include <model.h>
```

DESCRIPTION

There are certain inevitable distinctions between HP-UX implementations due to hardware differences. Where such distinctions exist, conditional compilation or other definitions can be used to isolate the differences. Flags and typedefs to resolve these distinctions are collected in the **<model.h>** header file which contains constants identifying various HP-UX implementations.

For example, header file **model.h** contains the following constants whose values are defined in **<sys/magic.h>**:

```
#define   HP_S_500    HP9000_ID
#define   HP_S_200    HP98x6_ID
#define   HP_S_300    CPU_HP_MC68020
#define   HP_S_800    CPU_PA_RISC1_0
#define   HP_S_700    CPU_PA_RISC1_1
```

Other such constants are added as appropriate when HP-UX extends to other machines in subsequent releases.

In addition, **model.h** has a statement defining the preprocessor constant **MYSYS** to represent the specific implementation for which compilation is desired. **MYSYS** is always equal to one of the constants above.

Conditional compilation can be used to adapt a single file for execution on more than one HP-UX implementation if the file contains implementation- or architecture-dependent features. For example, the code segment:

```
#if MYSYS==HP_S_400
      <statements>
#endif
```

causes statements following the if statement to be compiled *only* if the system processor is an HP 9000 Series 400 machine.

model.h also contains typedefs for several predefined types to enhance portability of certain types of code and files.

int8, u_int8	Signed and unsigned 8-bit integers.
int16, u_int16	Signed and unsigned 16-bit integers.
int32, u_int32	Signed and unsigned 32-bit integers.
machptr, u_machptr	Signed and unsigned integers large enough to hold a pointer.

Certain C preprocessor conditional compilation variables are defined to aid in implementation-dependent code. See *cpp*(1).

SEE ALSO

cc(1), cpp(1), magic(4).

NAME

netgroup - list of network groups

DESCRIPTION

File `/etc/netgroup` defines network-wide groups, and is used for permission checking when executing remote mounts, remote logins, and remote shells. For remote mounts, the information in `netgroup` classifies machines; for remote logins and remote shells, it classifies users. Each line of the `netgroup` file defines a group and has the format

groupname member1 member2 ...

where member*i* is either another group name, or a triple.

(*hostname, username, domainname*)

If any of these three fields are left empty, it signifies a wild card. Thus

```
universal (,,)
```

defines a group to which everyone belongs. Field names that begin with something other than a letter, digit or underscore (such as -) do not match any value. For example, consider the following entries.

```
justmachines      (analytica,-,YOURDOMAIN)
justpeople        (-,root,YOURDOMAIN)
```

Machine `analytica` belongs to the group `justmachines` in the domain `YOURDOMAIN`, but no users belong to it. Similarly, the user `root` belongs to the group `justpeople` in the domain `YOURDOMAIN`, but no machines belong to it.

Note, the domain name field must match the current domain name (as returned by the `domainname` command), or the entry is not matched. Also, the user-name field is ignored for remote mounts. Only the hostname and domainname are used.

The Network Information Service (NIS) can serve network groups. When so used, they are stored in the following NIS maps.

netgroup
netgroup.byuser
netgroup.byhost

Refer to *ypserv*(1M) and *ypfiles*(4) for an overview of Network Information Service.

AUTHOR

`netgroup` was developed by Sun Microsystems, Inc.

FILES

`/etc/netgroup`

SEE ALSO

makedbm(1M), mountd(1M), ypmake(1M), ypserv(1M), getnetgrent(3C), hosts.equiv(4), ypfiles(4).

Installing and Administering NFS Services, Chapter 7: NIS Configuration.

NAME

netrc - login information for *ftp*(1) and *rexec*(3N)

DESCRIPTION

The `.netrc` file contains login and initialization information used by the `ftp` auto-login process, by the `rexec()` library routine, and by the `rexec` command (see *ftp*(1), *rexec*(3N), and *remsh*(1)), respecively. This file is optional. It exists, if at all, in the user's home directory.

If the `.netrc` file contains password or account information for use other than for anonymous FTP, its owner must match the effective user ID of the current process. Its read, write, and execute mode bits for group and other must all be zero, and it must be readable by its owner. Otherwise, the file is ignored.

The file can contain the following tokens, separated by spaces, tabs, or new-lines.

`machine` *name* Identify a remote machine name. The auto-login process searches the `.netrc` file for a `machine` token that matches the remote machine specified on the `ftp` command line, as an ftp `open` command argument, or as the **ahost* parameter to `rexec()`. Once a match is made, the subsequent `.netrc` tokens are processed, stopping when the end of file is reached or another `machine` token or a `default` token is encountered.

`default` Same as `machine` *name* except that `default` matches any name. There can be only one `default` token, and it must be after all `machine` tokens. This is normally used for ftp as follows:

```
default login anonymous password user@site
```

This provides automatic anonymous FTP login to machines not specified in `.netrc`. This can be overridden in ftp by using the `-n` flag to disable auto-login.

`login` *name* Identify a user on the remote machine. If this token is present, the `ftp` or `rexec()` auto-login process initiates a login using the specified name. If this token matches the user name used by the `rexec -l` command option, or by default the local user name), `rexec` uses the `password` token, if present.

`password` *string* Supply a password. If this token is present, the auto-login process supplies the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the `.netrc` file for any user other than `anonymous`, `ftp` aborts the auto-login process if the `.netrc` is readable by anyone other than the owner. Also note that the passwords in `.netrc` are not encrypted.

`account` *string* Supply an additional account password for `ftp` login. If this token is present, the auto-login process supplies the specified string if the remote server requires an additional account password, or the auto-login process initiates an `ACCT` command if it does not.

`macdef` *name* Define an `ftp` macro. This token is just like the ftp `macdef` command. A macro is defined with the specified name; its contents begin with the next `.netrc` line and continue until an empty line (consecutive new-line characters) is encountered. If a macro named `init` is defined, it is automatically executed as the last step in the `ftp` auto-login process.

EXAMPLES

The following is a valid entry for the host `hpxdzg` whose `guest` account has the password `sesame`:

```
machine  hpxdzg  login  guest  password  sesame
```

WARNINGS

It is a security risk to have unencrypted passwords in a file.

AUTHOR

netrc was developed by the University of California, Berkeley.

FILES

`$HOME/.netrc`

SEE ALSO
ftp(1), remsh(1), rexec(3N).

NAME

nettlgen.conf - Network Tracing and Logging configuration file

SYNOPSIS

`/etc/conf/nettlgen.conf`

DESCRIPTION

`/etc/conf/nettlgen.conf`, the configuration file for Common Network Tracing and Logging commands, contains configuration information used by the `nettl` and `netfmt` commands (see *netitl*(1M) and *netfmt*(1M)). Subsystems update this file during installation when the `update` utility runs their customize script (see *update*(1M)). The `nettlconf` command (see *nettlconf*(1M)) maintains subsystem data in this file, allowing subsystems to safely add, modify, or delete existing entries in the file.

The file is composed of records containing fields which are separated by colons (`:`). Each line is a unique record containing either global log information or subsystem information. The first field in each record is the tag field which identifies the type of information contained in that record. A `LOG` tag identifies log information; a `SS` tag identifies subsystem information. Blank lines or lines beginning with `#` are ignored.

Log Record

The log record defines static information used to configure logging defaults such as the name of the log file and whether to turn console logging on or off. Note that only the last log record encountered in the file is used; prior log records are ignored. Users can alter the log information to suit their particular needs. For the log information changes to take effect, the system administrator must stop and restart the tracing and logging facility using the `nettl` command.

Log record fields are as follows:

Field Number	Name	Description
1	tag	Contains **LOG** tag string.
2	Console Logging Flag	Set to 1 if console logging is to be enabled, 0 if not.
3	Log Port Size	Amount of space to reserve for internal log message buffers. Specified in Kbyte units.
4	Maximum Log File Space	Determines the maximum logging file space to be allowed. Specified in Kbyte units.
5	Log File prefix	Path and name of the log file, without the type and age extension (.LOG0x, where x is 0 or 1).
6	Console Filter File	Name of filter configuration file used for console logging.

Console logging is used to print log messages on the system console. The information that is displayed on the console is controlled by the configuration information contained in the console filter file. If there is no console present this feature can be turned off. If more information is desired than the special terse form used for console logging, turn off console logging and start a formatter with an options file specifying the filters to use.

The *Log Port Size* defines the number of outstanding messages possible in the log queue. For logging, 256-byte buffers are used. The number chosen here indicates how much space to allocate in kilobytes. The default size is 8192 bytes (specified by 8), which is split into thirty-two 256-byte blocks. The first block is reserved by the system, leaving 31 blocks for log messages. Each log message starts on a new block, taking 64 bytes of overhead. In addition, each block takes 8 bytes of overhead. The largest message that can be stored using the default size is 7624 bytes ((31 * 256) - (31 * 8) - 64). Most log messages are fairly small, so choosing 8K of buffer is sufficient for the logging facility to keep up with a large volume of messages.

The *Maximum Log File Space* determines the maximum logging file space to be allowed. Log files are split into two parts. When an individual log file reaches one-half of the maximum specified here, the logging

system deletes any existing old file, renames the current file to the old file, and starts a new file. The default specification allows for 1 Megabyte of total log file storage (each file does not exceed 500K bytes). Since logging is usually infrequent and log messages are fairly small, this should be more than adequate for all needs. The rate at which the file space fills up depends on what level of logging is turned on for each subsystem, the volume of traffic, frequency of connections, etc; and is very difficult to predict.

The *Console Filter File* specifies the name of the file containing formatter filters used for console logging. This file contains filters that control the logged information displayed on the console. The syntax of this file is the same as the filter configuration files that are used with the **netfmt** command. See *netfmt*(1M) for more details on filter configuration files.

If the console filter file does not exist the specified file is created with a default set of filters which will display DISASTER messages on the console. If the console filter file does exist and contains a *time_from* filter, the *time_of_day* and *day_of_year* fields in the filter will be updated every time **nettl** is started.

The *Console Filter File* field is optional. If omitted the default file **/usr/adm/conslog.opts** will be used.

Subsystem Record

The subsystem record defines the information for that subsystem, and has ten fields including the tag field. The fields are separated by colons (**:**); thus no field can contain a colon. An empty field can be represented by the string **NULL**. NOTE: the information in the subsystem records should only be changed by the subsystem's customization script using the **nettlconf** command. Users should not change this information unless directed by a Hewlett-Packard support representative.

Subsystem record fields are as follows:

Field Number	Name	Description
1	tag	Contains **SS** tag string.
2	Subsystem ID	An integer between 0 and 255. This number is set by the HP factory and must not be changed.
3	Subsystem Mnemonic	A text string consisting of letters, numbers, and the underscore character. The string is set at the factory and must not be changed.
4	Initial Log Class	Logging class for the subsystem when the tracing and logging facility is initialized. This is a numeric value as shown below.
5	Subsystem Type	Set to **k** if the subsystem exists in the kernel, **u** if not.
6	Subformatter Shared Library	Name of the shared library file containing the subformatter functions listed below.
7	Subformatter Message Catalog	Basename of the message catalog without the **.cat** extension and path prefix.
8	Subformatter Function	C function in the subformatter library to call when formatting data for this subsystem.
9	Subformatter Options	C function in the subformatter library to call to get filter options for this subsystem.
10	Group Name	A text string to be used in the header banner line in the formatted output.

The recommended setting for the default logging level is set by the products' customization scripts. It can be changed by the user if another level of logging is desired on initialization. The available classes are Disaster (8), Error (4), Warning (2), and Informative (1). Classes can be combined by adding the numbers; thus Disaster and Error together become 12. The logging level can also be changed at run time using the *nettl -log* command. Disaster class is always turned on, even if not specified in this configuration file; thus, specifying the value 14 or 6 turns on Disaster, Error and Warning.

If the subformatter library file name does not contain an absolute path, it is assumed to be under `/usr/lib`. The subformatter library *must* be a shared library.

EXTERNAL INFLUENCES

Message catalogs are found in the path determined by the environment variable **NLSPATH**. Default message catalogs are found in `/usr/lib/nls/%L/%N.cat` where the contents of the **LANG** environment variable is substituted for the *%L* field, and the name specified in this parameter is substituted for the *%N* field.

EXAMPLES

The following example shows the default logging information. Console logging is enabled; logging uses 8 Kbytes to hold log messages; the log files are limited to 1000 Kbytes total (500 Kbytes per file); the log files are `/usr/adm/nettl.LOG00` and `/usr/adm/nettl.LOG01`; and the console logging filter file is `/usr/adm/conslog.opts`. Most recent data is always in the `.LOG00` file.

```
#
# LOG INFORMATION
#

LOG:1:8:1000:/usr/adm/nettl:/usr/adm/conslog.opts
```

The following example turns off console logging, and limits the size of the log file space to 100 Kbytes. Other values are the same as the default.

```
#
# LOG INFORMATION
#

LOG:0:8:100:/usr/adm/nettl:/usr/adm/conslog.opts
```

The following example shows a typical subsystem record. These records should not be changed by the user, but are set by the subsystems using during the customize step of `update .`

```
#
# TEST SUBSYSTEMS
#

SS:96:TEST_ID_1:8:u:NULL:netfmt:subsys_GENERIC_format: \
ss_96_go:FORMATTER
SS:97:TEST_ID_2:8:u:NULL:netfmt:subsys_GENERIC_format: \
ss_97_go:FORMATTER
```

`Note:` The continuation marks in this example (\ at end-of-line) and the following one are placed for readability purposes only. `nettl` and `netfmt` do not understand continuation marks.

The following entry *must always* be included in the configuration file. This defines the subsystem for the formatter itself; if it is not in the file, the formatter will not operate properly.

```
#
# FORMATTER SUBSYSTEMS
#

SS:127:FORMATTER:12:u:NULL:netfmt:subsys_GENERIC_format: \
subsys_127_get_options:FORMATTER
```

FILES

/etc/conf/nettlgen.conf

SEE ALSO

netfmt(1M), nettl(1M), nettlconf(1M), update(1M).

NAME

networks - network name data base

DESCRIPTION

The `/etc/networks` file associates Internet addresses with official network names and aliases. This allows the user to refer to a network by a symbolic name instead of using an internet address. For each network, a single line should be present with the following information:

<official network name> <network number> <aliases>

Aliases are other names under which a network is known. For example:

```
loop  192.46.4  testlan
```

where the network named `loop` is also called `testlan`.

A line cannot start with a blank (tab or space character). Items are separated by any number or combination of blanks. A # character indicates the beginning of a comment. Characters from the # up to the end of the line are not interpreted by routines which search the file. Trailing blanks are allowed at the end of a line. For the DARPA Internet network, this file is normally created from the official network database maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up-to-date regarding unofficial aliases and/or unknown networks.

Network numbers can be specified in conventional Internet dot notation using the `inet_network()` routine from the internet address manipulation library (see *inet*(3N). Network names can contain any printable character other than a white space, new-line, or comment character.

EXAMPLES

See `/etc/networks`.

AUTHOR

`networks` was developed by the University of California, Berkeley.

FILES

`/etc/networks`

SEE ALSO

getnetent(3N).

NAME

nlist - nlist structure format

SYNOPSIS

```
#include <nlist.h>
```

REMARKS

The exact content of the structure defined below can be best found by examining `/usr/include/nlist.h`. It varies somewhat between various HP-UX implementations.

DESCRIPTION

`nlist()` can be used to extract information from the symbol table in an object file (see *nlist*(3C)). Since symbol tables are machine dependent (as defined in each implementation's copy of `<a.out.h>`), a header file, `nlist.h` is defined to encapsulate the differences.

The `nlist()` function, when used with the `nlist` structure can be used to extract certain information about selected symbols in the symbol table. The data associated with each symbol is machine specific, thus only the name and position of the `n_name` field in the `nlist` structure is standardized by HP-UX. The rest of the structure includes at least the value and type of the symbol. The names and meanings of all fields not standardized will change no more than necessary.

```
struct nlist {
    char    *n_name;
    /* other fields as needed;
       the following are suggested if they apply */
    long            n_value;
    unsigned char   n_type;
    unsigned char   n_length;
    short           n_unit;
    short           n_sdindex;
};
```

SEE ALSO

nlist(3C), a.out(4).

NAME

passwd - password file, pwd.h

DESCRIPTION

`passwd` contains the following information for each user:

- login name
- encrypted password
- numerical user ID
- numerical group ID
- reserved field, which can be used for identification
- initial working directory
- program to use as shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. Each user is separated from the next by a new-line. This file resides in the `/etc` directory. It can and does have general read permission and can be used, for example, to map numerical user IDs to names If the password field is null and `/.secure/etc/passwd` does not exist, no password is demanded.

If the shell field is null, `/bin/sh` is used.

The system ignores the encrypted password field in `/etc/passwd` entries. The encrypted password consists of 13 characters chosen from a 64-character set of "digits" described below, except when the password is null, in which case the encrypted password is also null. Login can be prevented by entering in the password field a character that is not part of the set of digits (such as `*`).

The characters used to represent "digits" are `.` for 0, `/` for 1, `0` through `9` for 2 through 11, `A` through `Z` for 12 through 37, and `a` through `z` for 38 through 63.

Password aging is put in effect for a particular user if his encrypted password in the password file is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced in the first instance by the super-user.) This string defines the "age" needed to implement password aging.

The first character of the age, *M*, denotes the maximum number of weeks for which a password is valid. A user who attempts to login after his password has expired is forced to supply a new one. The next character, *m*, denotes the minimum period in weeks that must expire before the password can be changed. The remaining characters define the week (counted from the beginning of 1970) when the password was last changed (a null string is equivalent to zero). *M* and *m* have numerical values in the range 0 through 63 that correspond to the 64-character set of "digits" shown above. If $m = M = 0$ (derived from the string `.` or `..`) the user is forced to change his password next time he logs in (and the "age" disappears from his entry in the password file). If $m > M$ (signified, for example, by the string `./`) only the super-user can change the password.

getpwent(3C) designates values to the fields in the following structure declared in <`pwd.h`>:

```
struct passwd {
    char    *pw_name;
    char    *pw_passwd;
    int     pw_uid;
    int     pw_gid;
    char    *pw_age;
    char    *pw_comment;
    char    *pw_gecos;
    char    *pw_dir;
    char    *pw_shell;
    long    pw_audid;
    int     pw_audflg;
};
```

It is suggested that the range 0-99 not be used for user and group IDs (`pw_uid` and `pw_gid` in the above structure) so that IDs that might be assigned for system software do not conflict.

The user's full name, office location, extension, and home phone stored in the `pw_gecos` field of the `passwd` structure can be set by use of the `chfn` command (see *chfn*(1)) and is used by the *finger*(1)

command. These two commands assume the information in this field is in the order listed above. A portion of the user's real name can be represented in the `pw_gecos` field by an `&` character, which some utilities (including `finger`) expand by substituting the login name for it and shifting the first letter of the login name to uppercase.

SECURITY FEATURES

A second password file, `/.secure/etc/passwd` maintains encrypted passwords on the system and prevents users from viewing them. The `/.secure/etc/passwd` file contains for each user the following information:

- login name
- encrypted password
- numerical audit ID
- numerical audit flag

Like `/etc/passwd`, `/.secure/etc/passwd` is an ASCII file. Fields within each user's entry are separated by colons. When it exists on the system, `/.secure/etc/passwd` contains the encrypted passwords to prevent access by non-privileged users.

The passwords contained in `/.secure/etc/passwd` take precedence over those contained in the encrypted password field of `/etc/passwd`. User authentication is done using the encrypted passwords in this file. The password aging mechanism described above also applies to `/.secure/etc/passwd`.

The `pw_audid` and `pw_audflg` also reside in `/.secure/etc/passwd`.

getpwent(3C) designates values to the fields in the following structure, which is declared in <`pwd.h`>:

```
struct s_passwd {
    char    *pw_name;
    char    *pw_passwd;
    long    pw_audid;
    int     pw_audflg;
};
```

NETWORKING FEATURES

NFS

The `passwd` file can have entries that begin with a plus (+) or minus (-) sign in the first column. Such lines are used to access the Network Information System network database. A line beginning with a plus (+) is used to incorporate entries from the Network Information System. There are three styles of + entries:

+	Insert the entire contents of the Network Information System password file at that point;
+*name*	Insert the entry (if any) for *name* from the Network Information System at that point
+@*name*	Insert the entries for all members of the network group *name* at that point.

If a + entry has a non-null password, directory, gecos, or shell field, they override what is contained in the Network Information System. The numerical user ID and group ID fields cannot be overridden.

The *passwd* file can also have lines beginning with a minus (-), which disallow entries from the Network Information System. There are two styles of `- entries:`

-*name*	Disallow any subsequent entries (if any) for *name*.
-@*name*	Disallow any subsequent entries for all members of the network group *name*.

WARNINGS

User ID (uid) 17 is reserved for the Pascal Language operating system. User ID (uid) 18 is reserved for the BASIC Language operating system. These are operating systems for Series 300 and 400 computers that can co-exist with HP-UX on the same disk. Using these uids for other purposes may inhibit file transfer and sharing.

The information kept in the `pw_gecos` field may conflict with unsupported or future uses of this field. Use of the `pw_gecos` field for keeping user identification information has not been formalized within any of the industry standards. The current use of this field is derived from its use within the Berkeley Software

Distribution. Future standards may define this field for other purposes.

The following fields have character limitations as noted:

- Login name field can be no longer than 8 characters;
- Initial working directory field can be no longer than 63 characters;
- Program field can be no longer than 44 characters.
- Results are unpredictable if these fields are longer than the limits specified above.

The following fields have numerical limitations as noted:

- The user ID is an integer value between –2 and **UID_MAX** inclusive.
- The group ID is an integer value between 0 and **UID_MAX** inclusive.
- If either of these values are out of range, the *getpwent*(3C) functions reset the ID value to (**UID_MAX**).

EXAMPLES

NFS Example

Here is a sample **/etc/passwd** file:

```
root:3Km/o4Cyq84Xc:0:10:System Administrator:/:/bin/sh
joeuser:r4hRJr4GJ4CqE:100:50:Joe User,Post 4A,12345:/users/joeuser:/bin/ksh
+john:
-bob:
+@documentation:no-login:
-@marketing:
+:::Guest
```

In this example, there are specific entries for users **root** and **joeuser**, in case the Network Information System are out of order.

- User **john**'s password entry in the Network Information System is incorporated without change.
- Any subsequent entries for user **bob** are ignored.
- The password field for anyone in the netgroup **documentation** is disabled.
- Users in netgroup **marketing** are not returned by *getpwent*(3C) and thus are not allowed to log in.
- Anyone else can log in with their usual password, shell, and home directory, but with a **pw_gecos** field of **Guest**.

NFS Warnings

The plus (+) and minus (-) features are NFS functionality; therefore, if NFS is not installed, they do not work. Also, these features work only with **/etc/passwd**, but not with **/.secure/etc/passwd**. When **/.secure/etc/passwd** is installed, the encrypted passwords can be accessed only in **/.secure/etc/passwd**. Any user entry in the Network Information System database also must have an entry in **/.secure/etc/passwd**.

The uid of –2 is reserved for remote root access by means of NFS. The **pw_name** usually given to this uid is **nobody**. Since uids are stored as unsigned values, the following define is included in <**pwd.h**> to match the user **nobody**.

```
UID_NOBODY  ((ushort) 0xfffe)
```

FILES

/etc/passwd

SEE ALSO

chfn(1), finger(1), login(1), passwd(1), a64l(3C), crypt(3C), getpwent(3C), limits(5).

STANDARDS CONFORMANCE

passwd: SVID2, XPG2

NAME

pcf -port configuration file, used by DDFA software

Description

This file is used by the **HP Datacommunications and Terminal Controller** (DTC) **Device File Access** (DDFA) software to configure individual DTC ports. `pcf` is the generic name of the template file. In practice it is renamed for each port that needs different configuration values, and the values are altered appropriately for the device attached to the port. The `pcf` is referenced by an entry in the Dedicated Ports file (`dp`). The Dedicated Port Parser (`dpp`) parses the `dp` file and calls the Outbound Connection Daemon (`ocd`) program to spawn a daemon for each valid line in the `dp` file. A valid line is one in which the fourth field is the name of a `pcf`.

The master `pcf` is `/etc/newconfig/ddfa/pcf`, and should only be referenced in the `dp` file if the default values it contains are correct for the ports. If different values are needed, `/etc/netconfig/ddfa/pcf` should be copied to another directory and the copy should be modified and referenced in `dp`. The recommended procedure is to create the directory `/etc/ddfa` to hold the pcfs and the modified dp file.

Refer to *ddfa*(7) for information on how to configure and install the DDFA software.

The file consists of the names of variables and their values. The variables are shown terminated by a colon (`:`), but this is not mandatory. A variable and its value can be separated by spaces or tabs. Only one variable-value pair is allowed per line. Only the value should be altered; the variable name should not be changed.

The file contains the following information:

`telnet_mode:` This can have the value `disable` or `enable`. When it is enabled, data transfer over the network use the Telnet protocol. This option *must* be enabled for the DTC.

`timing_mark:` This can have the value `enable` or `disable`. When it is enabled, a telnet timing-mark negotiation is sent to the DTC after all user data has been transferred. `ocd` waits for a reply to the timing mark negotiation before closing the connection. This ensures that all data has been output from the DTC buffers to the device before the buffers are flushed. It should therefore be enabled for the DTC.

`telnet_timer:` This defines the time, in seconds, during which the software waits for a response to the telnet timing mark and binary negotiation. If the timer expires, an error message is logged to `/usr/adm/syslog` and the error is transmitted to the user application.

`binary_mode:` This can have the value `disable` or `enable`. When it is enabled, data transfer over the network is in binary mode, and treatment of special characters (such as XON/XOFF) is disabled.

Due to the absence of flow control, data integrity cannot be guaranteed when `binary_mode` is enabled.

Note that even if `binary_mode` is disabled, it can be negotiated at any time by the application setting `IXON` to `0` in the `termio` data structure.

`open_tries:` This defines the number of times the software tries to open a connection before giving up. If the value is `0` the software tries "forever" (approximately 68 years). If the retry process fails, an error message is logged to `/usr/adm/syslog`. The error message is also transmitted to the user application.

The retry process can be interrupted by sending the `SIGUSR2` signal to the ocd process using `kill -17` *pid*.

Note that if the application exits after asking `ocd` to open the connection to the DTC, `ocd` continues trying to open until `open_tries` and/or `open_timer` are exceeded.

`open_timer:` This defines the time in seconds between tries. If the value is `0`, `ocd` uses an exponential retry period algorithm up to 32 seconds; i.e., 1 2 4 8 16 32 32 32 ...

`close_timer:` This defines the time in seconds between the close call made by the application on the pty slave and the moment when the connection is actually closed. Setting this value to, for example, 5 seconds avoids the overhead of opening and closing the connection when a spooler spools several files at a time. Setting a sufficiently high value effectively leaves the connection permanently open.

`status_request:`
This can have the value **disable** or **enable**. When it is enabled, the software sends a status request to the device attached to the server and processes the reply as follows:

LP_OK *(0x30)* **ocd** continues processing.

LP_NO_PAPER *(0x31)*
ocd retries within the limits of the status timer.

LP_BUSY *(0x32)*
ocd retries within the limits of the status timer.

LP_OFF_LINE *(0x23)*
ocd retries within the limits of the status timer.

LP_DATA_ERROR *(0x38)*
ocd retries within the limits of the status timer.

`status_timer:` This defines the time, in seconds, after which the software no longer waits for the reply to the status request. If the timer expires, an error message is logged to **/usr/adm/syslog**. The error condition is also transmitted to the user application.

`eight_bit:` This can have the value **enable** or **disable**.

Normally, data bytes processed by the pty have bit 7 stripped. If **eight_bit** is enabled, the stripping is disabled. If eight_bit is disabled, stripping is enabled, and bit 7 is stripped. This can also be achieved by changing the pseudonym's termio structure using ioctl() commands.

`tcp_nodelay:` This can have the value **enable** or **disable**.

If it is enabled, data is sent to the LAN as it is received. It can be disabled if the software is sending packets faster than the server can accept.

The default values are:

```
telnet_mode     enable
timing_mark     enable
telnet_timer    120
binary_mode     disable
open_tries      1500
open_timer      30
close_timer     0
status_request  disable
status_timer    30
eight_bit       disable
tcp_nodelay     enable
```

FILES

```
/etc/dpp
/etc/ocdbug
/etc/ocd
/etc/dpp_login.bin
/etc/utmp.dfa
/etc/newconfig/ddfa/pcf
/etc/newconfig/ddfa/dp
```

SEE ALSO

ddfa(7) dp(4) dpp(1m) ocd(1m) ocdebug(1m).

NAME

pdf - Product Description File

DESCRIPTION

A **Product Description File** describes product files contained in the HP-UX operating system. It consists of a file containing a single line entry for each file described, where each entry contains the following fields:

pathname
owner
group
mode
size
links
version
checksum
linked_to

Fields are separated by a colon (:), and contain the information indicated:

pathname Absolute pathname of the file (starts with /). If pathname is preceded by ?, it is an optional file that may or may not be present on the system.

owner Symbolic or numeric ID of the owner of the file.

group Symbolic or numeric ID of the group of the file.

mode Symbolic representation of file type and permission information as displayed by the `ls -l` command.

size Size of the file in bytes. In the case of device special files, it is the major/minor number. Directory sizes are not recorded.

links Number of hard links to *pathname*.

version Numeric value of the revision of the file. Commands supporting PDFs determine this value by invoking the `what` command on the file and searching for a revision number (see *what*(1)). If no revision is found, `ident` invoked (see *ident*(1)). The version number recorded is the first one encountered. If no version number is found, the field is empty.

checksum Result of the application of the Ethernet (and hence IEEE 802.3) CRC checksum algorithm to the file's contents.

linked_to File to which *pathname* is linked, whether with a **hard** or **symbolic** link. If *pathname* is not a link, this field is empty.

Some commands (namely `pdfdiff` and `pdfck`) rely on the convention that one file in a set of hard links is considered the primary file, indicating no *linked_to* file in the PDF, while the remaining files in the set all indicate the primary file as the *linked_to* (see *pdfdiff*(1M) and *pdfck*(1M)). This convention prevents double counting in size calculations, and allows some efficiencies in algorithms for checking consistency of links.

Empty fields indicate a *"don't care"* status. Any field except *pathname* can be empty.

comment lines in the file begin with the percent character (%). The first line of the file is always the comment:

```
% Product Description File
```

The second comment line is produced by the `mkpdf` command's `-c` option. For HP-UX files, this comment usually indicates the product name and release.

EXAMPLE

Here is an example product desription file:

```
% Product Description File
% fileset TEST, Release 1.0
/bin/basename:bin:bin:-r-xr-xr-x:2244:1:66.2:4066520052:
/bin/cat:bin:bin:-r-xr-xr-x:4740:1:66.2:2516588651:
/bin/cc:bin:bin:-r-xr-xr-x:24576:2:66.12:330130894:
```

```
/bin/dirname:bin:bin:-r-xr-xr-x:1936:1:64.3:549465715:
/bin/grep:bin:bin:-r-xr-xr-x:11988:3:66.11:2104745188:
/bin/ls:bin:bin:-r-xr-xr-x:24576:6:66.3:312786007:
/bin/ll:::::6:::/bin/ls
/bin/su:root:bin:-r-sr-xr-x:90112:1:66.2:3088851439:
% total size is 160172 bytes.
% total size is 158 blocks.
```

WARNINGS

The checksum algorithm is different than that used by the 7.0 Release version of the commands.

AUTHOR

The specification of PDF is derived from an early draft proposal for *Bill of Materials* in *IEEE POSIX P1003.2* (Draft 2). This proposal was later dropped from the standard. The implementation is by HP.

FILES

`/system/*/pdf`

SEE ALSO

mkpdf(1m), pdfdiff(1m), pdfck(1m).

NAME

ppl.ipool - pool of local Internet addresses

DESCRIPTION

Local addresses for serial connections can be assigned in one of two ways:

- Local administrator specifies the IP address in file `ppl.remotes`.
- Local administrator does not specify the local address in file `ppl.remotes`, but rather leaves that field blank and specifies a pool of IP addresses to be allocated for serial connections. In this case `ppl` chooses the first unallocated address from that pool and uses it for the local address (see *ppl*(1)).

The `ppl.ipool` file contains Internet addresses in dotted decimal format, one address per line. Lines that begin with a # are ignored.

FILES

`/usr/lib/ppl/ppl.ipool`	full path name of `ppl.ipool` file

AUTHOR

`ppl.ipool` was developed by HP.

SEE ALSO

ppl(1).

NAME

ppl.remotes - ppl configuration information for remote hosts

DESCRIPTION

The `ppl.remotes` file contains one entry for each remote host that `ppl` can communicate with (see *ppl*(1)). Lines that are blank or begin with a `#` in column 1 are ignored. Each entry is a multi-line *form*. Each line of this form contains space for variable data and a fixed text string describing what data goes on the line. *All lines in the form must be present and undamaged or they cannot be parsed correctly.*

The system administrator configures `ppl` operation by replicating the *form* and filling it in for each remote host. Depending on the desired configuration, some lines of the form do not apply and are left blank. Other non-critical fields can be left blank, leaving the `ppl` program free to choose an appropriate value. Entry syntax is specified in comments in the file. Here is a sample form for one remote host:

```
# sample dial in/out to a remote system

hewlett             # remote host name or Internet address
packard             # local host name or Internet address
                    # Internet mask
SLIP                # protocol [SLIP] [ASLIPC] [ASLIPS] [PPP]
DIALIN & DIALOUT    # type [DIRECT] [DIALIN] [DIALOUT] [DIALIN & DIALOUT]
piper               # UUCP system name
                    # line parity [EVEN] [ODD] [NONE]
9600                # line speed
                    # serial line
9=2495574           # phone number
                    # modem control available [YES] [NO]
ogin-@-ogin guest ssword: 1JayBird %     # log in info
""ppl\shewlett running                   # command name
```

In this example, users on system `hewlett` can either dial in to `packard` or users on `packard` can dial in to `hewlett`. When `ppl` is run, only the needed fields are used. For example, the phone number is only used when dialing out. When dialing out, UUCP system `piper` is used to select a compatible line.

Fields

`# remote host name or Internet address`
: Must be supplied. Can be in dotted decimal notation, or machine name. If a name is given, it is translated to a numeric form using `gethostent()` (see *gethostent*(3N)).

`# local host name or Internet address`
: Optional. If supplied, it can be in dotted decimal notation or machine name. If a name is given, it is translated to a numeric form via `gethostbyname()` (see *gethostbyname*(3N)). If this field is left blank, `ppl` dynamically assigns a unique name from the pool of available addresses specified in the `ppl.ipool` file (see *ppl.ipool*(4)).

`# Internet mask`
: Specifies the Internet network mask. If no mask is specified, a default netmask is derived from the local Internet address.

`# protocol [SLIP] [ASLIPC] [ASLIPS] [PPP]`
: Specifies which encapsulation protocol `ppl` should use. As of this writing, SLIP, ASLIPC and ASLIPS are supported. Use `ASLIPC` to specify ASLIP client mode operation, and `ASLIPS` to specify ASLIP server mode operation.

`# type [DIRECT] [DIALIN] [DIALOUT] [DIALIN & DIALOUT]`
: specifies what types of connections are permitted by `ppl`. If `DIRECT` is specified, neither `DIALIN` nor `DIALOUT` can be specified. A direct connection implies that modem control signals are not used. If the serial line is disconnected, the program cannot detect it. This is discussed further under the modem control field below. If `DIALIN` is specified, a dial-in user is permitted to run `ppl` over the line used to dial in on. In this case, `ppl` is invoked with the `-i` (default) option, and the user's tty is used for the `ppl` encapsulation protocol. If `DIALOUT` is specified,

    `ppl -o` *remote_host*

causes the local machine to dial up the remote machine and initiate a connection. If **DIALIN & DIALOUT** is specified, either DIALIN or DIALOUT connections are permitted. Note that some non-standard connection arrangements such as security dial-back systems may require alternate type specifications. See *Using Serial Line IP Protocols* manual for more details.

 # UUCP system name
If the name of the remote computer is specified differently for the Internet remote host and the UUCP system, the UUCP name can be supplied. If this is left blank, the name given in the **#remote host name/inet_addr** field is used to locate a UUCP system name for dialing information if a dial out is required.

 # line parity [EVEN] [ODD] [NONE]
If this field is supplied, it is used for setting the terminal options only when **ppl** enters the encapsulation protocol phase. It is not used to establish a connection. These parameters are set by UUCP, getty, or other means.

 # line speed
is optional and only applies to outgoing (**-o**) connections. If supplied, this is used to set the rate on the tty line. If a UUCP line is used, this line provides the selection criteria when **ppl** selects a UUCP line. Most UUCP installations require that a speed be specified here in order to properly select a line. Line speed must be specified if a phone number or serial line is specified. Any baud rate supported by the system hardware can be used.

 # serial line
Specifies which tty device is to be used for outgoing connections. If not supplied, the UUCP **Systems** file is used to select one based on the the UUCP system name or *remote host name* given in the **ppl.remotes** file entry. The path is ignored. For example, the names **/dev/tty0p4** and **tty0p4** are equivalent.

 # phone number
Supplied for use on DIALOUT. When the phone number is supplied, it is used. If the phone number is not supplied here, the phone number listed in the UUCP **Systems** file is used. The syntax of the phone number is the same as the UUCP **Systems** file. Location names may be used as prefixes as in UUCP These are translated via the UUCP **Dialcodes** file.

 # modem control available
Properly controls the use of modem signals during and after the dialing phase of **ppl**. During the dialing phase, this field is used to control dialing activity. The protocol phase uses this field and the *type* field described above. If the *type* field is **DIRECT**, and this field is **NO** (or blank), modem signals are ignored. For all other combinations of these two fields, the modem signal are monitored; i.e., hang up signal causes **ppl** to exit gracefully. Typical installations on direct lines will use: **DIRECT** *type* and **NO** (or blank) for *modem control*. For modem ports, typical installations will use: no **DIRECT** *type*, and **YES** for *modem control.*

 # log in info
Supplies the dialogue needed to log in on the remote machine. The syntax is identical to the UUCP **Systems** file. This includes a series of send/expect strings separated by spaces. If this field is blank, it is assumed that no login is needed.

 # command name
Supplies an optional dialogue needed to run a complementary protocol command on the remote machine. For remote HP-UX machines, this command is typically to invoke the **ppl** program on the remote machine. The syntax is identical to the UUCP **Systems** file.

FILES

/usr/lib/ppl/ppl.remotes	full path name of **ppl.remotes**

WARNINGS
LOGIN option in the **OPTIONS** field is not used.

AUTHOR
ppl.remotes was developed by HP.

SEE ALSO
pplstat(1), ppl(1), ppl.ipool(4), ppl.users(4), ppl.ptmp(4),

Using Serial Line IP Protocols,
UUCP tutorial in *Remote Access Users Guide*.

NAME

ppl.users - translate user login names to default remote host names

DESCRIPTION

There are two ways for `ppl` to determine the remote address, and hence the corresponding `ppl.remotes` information (see *ppl*(1)):

- Specify it on the command line.
- If not specified on the command line, use the invoking user's login name to determine the default remote hostname to use by means of this file.

`ppl.users` contains a series of login-name/remote-host pairs, one pair per line. The login name must appear in `/etc/passwd`. The remote host can be either in dotted decimal format or a name that is translatable by `gethostent()` (see *gethostent*(3N)). Blank lines or lines beginning with a `#` are ignored.

FILES

`/usr/lib/ppl/ppl.users` full path name `of ppl.users`

AUTHOR

`ppl.users` was developed by HP.

SEE ALSO

ppl(1).

NAME

privgrp - format of privileged values

SYNOPSIS

```
#include <sys/privgrp.h>
```

DESCRIPTION

`setprivgrp()` sets a mask of privileges, and `getprivgrp()` returns an array of structures giving privileged group assignments on a per-group-ID basis (see *setprivgrp*(2) and *getprivgrp*(2)). `<privgrp.h>` contains the constants and structures needed to deal with these system calls, and contains:

```
/*
 * Privileged group definitions --
 * the numeric values may vary between implementations.
 */
#define PRIV_RTPRIO             1
#define PRIV_MLOCK              2
#define PRIV_CHOWN              3
#define PRIV_LOCKRDONLY         4
#define PRIV_SETRUGID           5

/* Maximum number of privileged groups in system */
#define PRIV_MAXGRPS           32

/*
.C "* Size of the privilege mask,"
.C "* based on largest numbered privilege"
.C "*/"
#define PRIV_MASKSIZ            1

/*
 * Structure defining the privilege mask
 */
struct privgrp_map {
    int             priv_groupno;
    unsigned int    priv_mask[PRIV_MASKSIZ];
};
```

Privileges are as follows:

`PRIV_RTPRIO`	Allows access to the `rtprio()` system call (see *rtprio*(2)).
`PRIV_MLOCK`	Allows access to the `plock()` system call (see *plock*(2)).
`PRIV_CHOWN`	Allows access to the `chown()` system calls (see *chown*(2)).
`PRIV_LOCKRDONLY`	Permits the use of the `lockf()` system call for setting locks on files open for reading only (see *lockf*(2)).
`PRIV_SETRUGID`	Permits the use of the `setuid()` and `setgid()` system calls for changing respectively the real user ID and real group ID of a process (see *setuid*(2) and *setgid*(2)).

Privileges are described in a multi-word mask. The value of the `#define` for each privilege is interpreted as a bit index (counting from 1). Thus a group-id can have several different privileges associated with it by having different bits ORed into the mask.

The system is configured with a specified maximum number of groups with special privileges. `PRIV_MAXGRPS` defines this maximum. Of this maximum, one is reserved for global privileges (granted to all processes), and the remainder can be assigned to actual group-ids.

`PRIV_MASKSIZ` defines the size of the multi-word mask used in defining privileges associated with a group-ID.

Privileges are returned to the user from the `getprivgrp()` system call in an array of structures of type `struct privgrp_map`. The structure associates a multi-word mask with a group-ID.

SEE ALSO
getprivgrp(2).

NAME

profile - set up user's environment at login time

DESCRIPTION

If the file `/etc/profile` exists, it is executed by the shell for every user who logs in. The file `/etc/profile` should be set up to do only those things that are desirable for *every* user on the system, or to set reasonable defaults. If a user's login (home) directory contains a file named `.profile`, that file is executed (via the shell's `exec .profile`) before the session begins. `.profile` files are useful for setting various environment parameters, setting terminal modes, or overriding some or all of the results of executing `/etc/profile`.

The following example is typical (except for the comments):

```
#  Make some environment variables global
    export MAIL PATH TERM
#  Set file creation mask
    umask 22
#  Tell me when new mail comes in
    MAIL=/usr/mail/myname
#  Add my /bin directory to the shell search sequence
    PATH=$PATH:$HOME/bin
# Set terminal type
    echo "terminal: \c"
    read TERM
    case $TERM in
     300)       stty cr2 nl0 tabs; tabs;;
     300s)      stty cr2 nl0 tabs; tabs;;
     450)       stty cr2 nl0 tabs; tabs;;
     hp)        stty cr0 nl0 tabs; tabs;;
     745|735)   stty cr1 nl] -tabs; TERM=745;;
     43)        stty cr1 nl0 -tabs;;
     4014|tek)  stty cr0 nl0 -tabs ff1; TERM=4014; echo "33;";;
     *)         echo "$TERM unknown";;
    esac
```

A more complete model `.profile` can be found in `/etc/d.profile`.

FILES

`$HOME/.profile`
`/etc/profile`

SEE ALSO

env(1), login(1), mail(1), sh(1), stty(1), su(1), environ(5), term(5).

NAME

proto - prototype job file for *at*(1)

SYNOPSIS

`/usr/lib/cron/.proto`
`/usr/lib/cron/.proto.` *queue*

DESCRIPTION

When a job is submitted to `at` or `batch`, the job is constructed as a Bourne shell script (see *at*(1) and *batch*(1)). The job file is created in `/usr/spool/cron/atjobs` as follows:

- `at` creates a header describing the job as an `at` job or a `batch` job. `at` jobs submitted to all queues other than queue `a` are listed as `batch` jobs. The header is:

  `: at job` for an `at` job, or

  `: batch job` for a `batch` job.

- A set of Bourne shell commands is added to make the environment (see *environ*(5)) for the `at` job the same as the current environment.

- `at` then copies text from the prototype file to the job file, except for special *variables* that are replaced by other text:

  `$d` Replaced by the current working directory.

  `$l` Replaced by the current file size limit (see *ulimit*(2)).

  `$m` Replaced by the current umask (see *umask*(2)).

  `$t` Replaced by the time at which the job should be run, expressed as seconds since January 1, 1970, 00:00 Coordinated Universal Time, preceded by a colon.

  `$<` Replaced by text read by `at` from the standard input (that is, the commands provided to `at` to be run in the job).

- When a job is submitted to queue *queue*, `at` uses the file `/usr/lib/cron/.proto.`*queue* as the prototype file if it exists. Otherwise it uses the file `/usr/lib/cron/.proto`.

EXAMPLES

The following `.proto` file creates commands to change the current directory, file size limit, and umask in the job to their respective values as they existed when `at` was originally run. These commands are inserted before the commands in the job:

```
cd $d
ulimit $l
umask $m
$<
```

SEE ALSO

at(1), queuedefs(4).

STANDARDS CONFORMANCE

`proto`: SVID2

NAME

protocols - protocol name data base

DESCRIPTION

This file associates protocol numbers with official protocol names and aliases. This allows the user to refer to a protocol by a symbolic name instead of a number. For each protocol a single line should be present with the following information:

<official protocol name> <official protocol number>

These mappings are defined in RFC 1010 *Assigned Numbers.*

Aliases are other names under which the protocol is also known. For example:

```
tcp      6        TCP
```

In this example, the library call `getprotobyname()` can be invoked as:

```
p = getprotobyname("TCP");
```

instead of

```
p = getprotobyname("tcp");
```

Both produce the same results.

A line cannot start with a space. Items are separated by any number of blanks and/or tab characters. A # character indicates the beginning of a comment. Characters from the # to the end of the line are not interpreted by routines which search the file.

Protocol names can contain any printable character other than a white space, new-line, or comment character. Trailing blanks or tabs are allowed at the end of a line.

EXAMPLES

```
tcp      6        TCP      # transmission control protocol
udp      17       UDP      # user datagram protocol
```

AUTHOR

`protocols` was developed by the University of California, Berkeley.

SEE ALSO

getprotoent(3N).

NAME

ptmp - ptmp entry format

DESCRIPTION

The `ppl` program updates file `ptmp` which holds user connection information (see *ppl*(1)). The `ptmp` file is similar in function and use to *utmp*(4). There is one entry for each tty device that `ppl` uses. If an entry does not exist for a tty, `ppl` creates a new entry at the end of the file; otherwise, it overwrites the old entry for that tty. When `ppl` exits, it sets the protocol field to 0, marking the entry idle. Idle entries indicate that, at some time in the past, an invocation of `ppl` was running on the tty.

The file is created initially by creating a zero-length file. If the file does not exist, `ppl` silently ignores this feature.

Each entry in the file has the following structure:

```
#define MAXSTRING 255
#define MAXINETADDR SOCK_ADDR_DATA_LEN   /* length of inet addrs */

struct ptmp {
    char log_name[MAXSTRING];          /* user login name */
    char rhost_name[MAXSTRING];        /* internet host name */
    char system_name[MAXSTRING];       /* uucp system name */
    char Linetaddr[MAXINETADDR];       /* local in dot format */
    char Rinetaddr[MAXINETADDR];       /* remote in dot format */
    char tty[MAXSTRING];               /* tty name: i.e. tty00 */
    int ni_unit;                       /* minor device i.e. ni0 */
    int status;                        /* current status of entry */
    long start_time;                   /* when protcol started */
};
```

FILES

`/usr/spool/ppl/ptmp` full path name of `ptmp`

AUTHOR

`ptmp` was developed by HP.

SEE ALSO

ppl(1), pplstat(1).

NAME

queuedefs - queue description file for at(1), batch(1), and crontab(1)

SYNOPSIS

`/usr/lib/cron/queuedefs`

DESCRIPTION

The `queuedefs` file describes the characteristics of the queues managed by `cron` (see *cron*(1M)). Each non-comment line in this file describes one queue. The format of the lines are as follows:

q.[*njob* `j`][*nice* `n`][*nwait* `w`]

The fields in this line are:

q The name of the queue, such that `a` is the default queue for jobs started by `at` (see *at*(1)), `b` is the queue for jobs started by `batch` (see *batch*(1)), and `c` is the queue for jobs run from a `crontab` file (see *crontab*(1)). Queue names `d` through `y` designate user-defined queues.

njob The maximum number of jobs that can be run simultaneously in that queue. Although any number can be specified here, the total number of jobs that can be run on all the queues is limited to 40.

nice The `nice` value to give to all jobs in that queue that are not run with a user ID of super-user (see *nice*(1)). The default value is 2.

nwait
The number of seconds to wait before rescheduling a job that was deferred because more than *njob* jobs were running in that job's queue, or because more than 40 jobs were running in all the queues (see *njob* above).

EXAMPLES

Consider the following `queuedefs` file:

```
a.4j1n
b.2j2n90w
```

The file is interpreted as follows:

`a.4j1n` The `a` queue, for `at` jobs (see `at (1))`, can have up to 4 jobs running simultaneously, and those jobs will be run with a `nice` value of 1.

Since no *nwait* value is given, if a job cannot be run because too many other jobs are running, `cron` will wait 60 seconds before trying again to run it (see *cron*(1M)).

`b.2j2n90w` The `b` queue, for `batch` jobs (see *batch*(1)), can have up to 2 jobs running simultaneously. Those jobs will be run with a `nice` value of 2. If a job cannot be run because too many other jobs are running, `cron` will wait 90 seconds before trying again to run it.

All other queues can have up to 100 jobs running simultaneously. They will be run with a `nice` value of 2, and if a job cannot be run because too many other jobs are running, `cron` will wait 60 seconds before trying again to run it.

SEE ALSO

at(1), batch(1), nice(1), crontab(1), cron(1M), proto(4)

STANDARDS CONFORMANCE

`queuedefs`: SVID2

NAME

ranlib - archive symbol table format for object libraries

SYNOPSIS

```
#include <ranlib.h>
```

DESCRIPTION

Any archive containing object files also includes an archive symbol table, thus allowing the linker (see *ld*(1)) to scan libraries in random (rather than sequential) order.

The archive symbol table (if it exists) is always the first file in the archive, but it is never listed. It is automatically created and/or updated by `ar` (see *ar*(1)).

The archive symbol table lists each externally known name in the archive, together with the offset of the archive element that defines that name. This offset is useful as an input argument to `lseek()` or `fseek()` (see *lseek*(2) and *fseek*(3S).

The archive symbol table file contains a header, a name pool of strings (the names of external symbols), and the archive symbol table. This allows for symbols with arbitrarily long names. The header contains a short integer which specifies the number of entries, and a long integer which specifies the size of the string table. Following this is the name pool. The last section of the file contains the archive symbol table entries. The structure of these entries is defined below:

```
    typedef long off_t;

struct    ranlib {
    union {
        off_t ran_strx;     /* string table index */
        char  *ran_name;
    } ran_un;
    off_t     ran_off;      /* lib member offset */
};
```

SEE ALSO

ar(1), ld(1), ar(4).

NAME

rcsfile - format of RCS files

DESCRIPTION

An RCS file is an ASCII file. Its contents are described by the grammar below. The text is free format, i.e., spaces, tabs and new-line characters have no significance except in strings. Strings are enclosed by "@". If a string contains the "@" symbol, the symbol must be doubled.

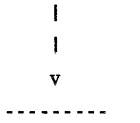
The meta syntax uses the following conventions: "|" (bar) separates alternatives; "{" and "}" enclose optional phrases; "{" and "}*" enclose phrases that may be repeated zero or more times; "{" and "}+" enclose phrases that must appear at least once and may be repeated; "<" and ">" enclose nonterminals.

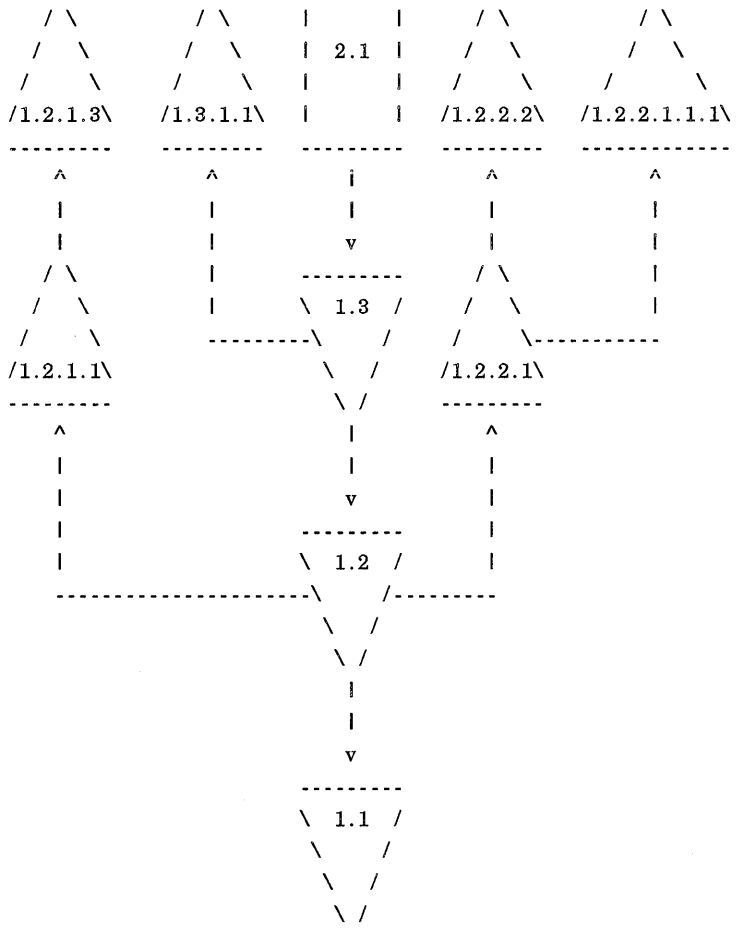
<rcstext>	::=	<admin> {<delta>}* <desc> {<deltatext>}*	
<admin>	::=	**head**	{<num>};
		access	{<id>}*;
		symbols	{<id> : <num>}*;
		locks	{<id> : <num>}*; {**strict** ;}
		comment	{<string>};
<delta>	::=	<num>	
		date	<num>;
		author	<id>;
		state	{<id>};
		branches	{<num>}*;
		next	{<num>};
<desc>	::=	**desc**	<string>
<deltatext>	::=	<num>	
		log	<string>
		text	<string>
<num>	::=	{<digit>{.}}+	
<digit>	::=	0 | 1 | ... | 9	
<id>	::=	<letter>{<idchar>}*	
<letter>	::=	A | B | ... | Z | a | b | ... | z	
<idchar>	::=	Any printing ASCII character except space, tab, carriage return, new line, and <special>.	
<special>	::=	; | : | , | @	
<string>	::=	@{any ASCII character, with "@" doubled}*@	

Identifiers are case sensitive. Keywords are in lowercase only. The sets of keywords and identifiers may overlap.

The <delta> nodes form a tree. All nodes whose numbers consist of a single pair (e.g., 2.3, 2.1, 1.3, etc.) are on the "trunk", and are linked through the "next" field in order of decreasing numbers. The "head" field in the <admin> node points to the head of that sequence (i.e., contains the highest pair).

All <delta> nodes whose numbers consist of 2n fields (n≥2) (e.g., 3.1.1.1, 2.1.2.2, etc.) are linked as follows. All nodes whose first (2n)-1 number fields are identical are linked through the "next" field in order of increasing numbers. For each such sequence, the <delta> node whose number is identical to the first 2(n-1) number fields of the deltas on that sequence is called the branchpoint. The "branches" field of a node contains a list of the numbers of the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers. Example:





WARNINGS

RCS is designed to be used with TEXT files only. Attempting to use RCS with non-text (binary) files results in data corruption.

AUTHOR

`rcsfile` was developed by Walter F. Tichy, Purdue University, West Lafayette, IN 47907.
Revision Number: 3.0; Release Date: 83/05/11.

SEE ALSO

ci(1), co(1), ident(1), rcs(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsintro(5).

NAME

resolver - resolver configuration file

SYNOPSIS

`/etc/resolv.conf`

DESCRIPTION

The **resolver** is a set of routines in the C library (see *resolver*(3N)) that provide access to the Internet Domain Name System. The resolver configuration file contains information that is read by the resolver routines the first time they are invoked by a process. The file is designed to be human-readable, and contains a list of keywords with values that provide various types of resolver information.

On a normally configured system this file should not be necessary. The only name server to be queried is on the local machine, the domain name is determined from the host name, and the domain search path is constructed from the domain name.

Recognized configuration options include:

`nameserver` Internet address (in dot notation) of a name server that the resolver should query. Up to **MAXNS** (currently 3) name servers can be listed, one per keyword. If there are multiple servers, the resolver library queries them in the order listed. If no `nameserver` entries are present, the default is to use the name server on the local machine. (The algorithm used is: Try a name server; if the query times out, try the next and continue until all name servers have been tried, then repeat trying all the name servers until a maximum number of retries have been made made).

`domain` Local domain name. Most queries for names within this domain can use short names relative to the local domain. If no `domain` entry is present, the domain is determined from the local host name returned by `gethostname()` (see *gethostname*(2)); the domain part is interpreted as everything after the first `.`. Finally, if the host name does not contain a domain part, the root domain is assumed.

`search` Search list for host-name lookup. The search list is normally determined from the local domain name; by default, it begins with the local domain name, then successive parent domains that have at least two components in their names. This can be changed by listing the desired domain search path following the `search` keyword with spaces or tabs separating the names. Most resolver queries will be attempted using each component of the search path in turn until a match is found. Note that this process may be slow and generates a lot of network traffic if the servers for the listed domains are not local, and that queries time out if no server is available for one of the domains.

The search list is currently limited to six domains with a total of 256 characters.

The first domain in the search list must be the local domain for short names to work properly in various files (such as `.rhosts` and `inetd.sec`)

The `domain` and `search` keywords are mutually exclusive. If more than one instance of these keywords is present, the last instance overrides.

The keyword and value must appear on a single line, and the keyword (e.g. `nameserver`) must start the line. The value follows the keyword, separated by white space.

Note that the resolver routine `res_init()` silently ignores errors when reading this file (see *res_init*(3N)).

EXAMPLES

A typical `resolv.conf` file resembles the following:

```
domain div.inc.com
nameserver 15.19.8.119
nameserver 15.19.8.197
```

AUTHOR

`resolver` was developed by the University of California, Berkeley.

FILES

`/etc/resolv.conf`	resolver configuration file

SEE ALSO

named(1m), resolver(3N), gethostent(3N), hostname(5)

NAME

rmtab - local file system mount statistics

DESCRIPTION

File `/etc/rmtab` contains a record of all clients that mounted remote file systems from this machine. Whenever a remote `mount` is done, an entry is made in the `rmtab` file of the machine serving that file system. `umount` removes the entry of a remotely mounted file system. `umount -a` broadcasts to all servers that they should remove all entries from `rmtab` created by the sender of the broadcast message. By placing a `umount -a` command in `/etc/brc`, `rmtab` tables are purged of entries made by a crashed host which, upon rebooting, did not remount the same file systems it had previously. The table is a series of lines of the following form:

hostname : *directory*

This table only preserves information between crashes, and is read only by `mountd` when it starts (see *mountd*(1M)). `mountd` keeps an in-core table to handle requests from commands such as `showmount` and `shutdown` (see *showmount*(1M) and *shutdown*(1M)).

WARNINGS

Although the `rmtab` table is close to the truth, it is not always totally accurate.

AUTHOR

`rmtab` was developed by Sun Microsystems, Inc.

FILES

`/etc/rmtab`

SEE ALSO

mount(1M), mountd(1M), showmount(1M), shutdown(1M), umount(1M).

NAME

rpc - rpc program number data base

SYNOPSIS

`/etc/rpc`

DESCRIPTION

File `/etc/rpc` contains user-readable names that can be used in place of RPC program numbers. Each line has the following information:

- Name of server for the RPC program
- RPC program number
- Aliases

Items are separated by any number of blanks and tab characters. A # anywhere in the file indicates a comment extending to the end of that line.

EXAMPLES

Here is an example of an `/etc/rpc` file:

```
#
# rpc 12.0 89/09/25
#
rstatd              100001        rstat rup perfmeter
rusersd             100002        rusers
nfs                 100003        nfsprog
ypserv              100004        ypprog
mountd              100005        mount showmount
ypbind              100007
walld               100008        rwall shutdown
yppasswdd           100009        yppasswd
etherstatd          100010        etherstat
rquotad             100011        rquotaprog quota rquota
sprayd              100012        spray
selection_svc       100015        selnsvc
dbsessionmgr        100016        unify netdbms dbms
rexd                100017        rex remote_exec
office_auto         100018        alice
```

AUTHOR

`rpc` was developed by Sun Microsystems, Inc.

FILES

`/etc/rpc`

SEE ALSO

getrpcent(3C).

NAME

sccsfile - format of SCCS file

DESCRIPTION

An SCCS file is an ASCII file consisting of six logical parts:

checksum	Sum of all characters in the file except the first line.
delta table	Contains information about each delta.
user names	Login names and/or numerical group IDs of users who are allowed to add deltas.
flags	Definitions of internal keywords. *comments* Arbitrary descriptive information about the file.
body	Actual text lines intermixed with control lines.

Throughout an SCCS file there are lines beginning with the ASCII **SOH** (start of heading) character (octal 001). This character is hereafter referred to as the *control character* and is represented graphically as **@**. Any line described below that is not depicted as beginning with the control character is prevented from beginning with the control character. All lines in the SCCS file are limited to **BUFSIZ** (defined in **<stdio.h>**) characters in length.

Entries of the form **DDDDD** represent a five-digit string (a number between 00000 and 99999).

The following describes each logical part of an SCCS file detail:

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

@DDD

The value of the checksum is the sum of all characters except those in the first line. The **@h** sequence provides a *magic number* consisting of the two bytes 0x01 and 0x68. (Other versions of UNIX-like operating systems usually use this same value but it may be displayed or documented as a single number with a different byte order.)

Delta table

The delta table consists of a variable number of entries of the form:

@s DDDDD/DDDDD/DDDDD
@d <type> <*SID*> yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
.
.
.
@c <comments> ...
.
.
.
@e

The first line (**@s**) contains the number of lines inserted/deleted/unchanged, respectively. The second line (**@d**) contains the type of the delta (currently, normal: **D**, and removed: **R**), the *SID* (SCCS ID) of the delta, the date and time when the delta was created, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The **@i**, **@x**, and **@g** lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

The **@m** lines (optional) each contain one **MR** number associated with the delta; the **@c** lines contain comments associated with the delta.

The **@e** line ends the delta table entry.

User names

The list of login names and/or numerical group IDs of users who are allowed to add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines **@u** and **@U**. An empty list allows anyone to make a delta. Any line starting with a ! prohibits the

specified group or user from making deltas.

Flags Keywords used internally (see *admin*(1) for more information on their use). Each flag line takes the form:

@f <flag> <optional text>

The following flags are defined:

@f t <type of program>
@f v <program name>
@f i <keyword string>
@f b
@f m <module name>
@f f <floor>
@f c <ceiling>
@f d <default-sid>
@f n
@f j
@f l <lock-releases>
@f q <user defined>
@f z <reserved for use in interfaces>

The above flags function as follows:

t Defines the replacement for the **%Y%** identification keyword.

v Controls prompting for **MR** numbers in addition to comments. If the optional text is present, it defines an **MR** number-validity checking program.

i Controls the warning/error aspect of the "No id keywords" message. When the **i** flag is not present, the message is only a warning; when the **i** flag is present, this message causes a fatal error (a *get* on the file fails, or the delta is not made).

b When the **b** flag is present, the **-b** keyletter can be used on the *get* command to cause a branch in the delta tree.

m Defines the first choice for the replacement text of the **%M%** identification keyword.

f Defines the "floor" release; the release below which no deltas can be added.

c Defines the "ceiling" release; the release above which no deltas can be added.

d Defines the default SID to be used when none is specified on a *get* command.

n Causes *delta* to insert a "null" delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (such as, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty.

j Causes *get* to allow concurrent edits of the same base SID. See *admin*(1) for restrictions.

l Defines a *list* of releases that are *locked* against editing (*get*(1) with the **-e** keyletter).

q Defines the replacement for the **%Q%** identification keyword.

z Used in certain specialized interface programs.

Comments Arbitrary text is surrounded by the bracketing lines **@t** and **@T**. The comments section typically contains a description of the file's purpose.

Body Consists of text lines and control lines. Text lines do not begin with the control character; control lines do. There are three kinds of control lines:

Type **Represented By:**

insert	**@I DDDDD**
delete	**@D DDDDD**
end	**@E DDDDD**

The digit string is the serial number corresponding to the delta for the control line.

WARNINGS

SCCS files can be any length, but the number of lines in the text file itself cannot exceed 99 999 lines.

SEE ALSO

admin(1), delta(1), get(1), prs(1).

NAME

sdf - structured directory format description

DESCRIPTION

SDF (Structured Directory Format) is the name given to the format of mounted media used by HP 9000 Series 500 HP-UX systems. This format is based upon the format used in Series 500 BASIC workstations.

Utilities listed under SEE ALSO below are provided for non-Series 500 access to SDF media. These utilities read and write data to and from SDF volumes, as well as retrieve information from an SDF volume.

The SDF utilities listed below are the only HP-UX utilities that recognize the internal contents of an SDF volume. To the rest of the HP-UX operating system, an SDF volume is simply a file containing unspecified data. Therefore, to access SDF media on any HP-UX system other than Series 500, **mount** cannot be used because the operating system cannot recognize it (see *mount*(1M)).

SDF file names are specified to the SDF utilities by concatenating the HP-UX path name for the SDF volume with the SDF file name, separating the two with a colon (**:**). For example,

```
/dev/rdsk/c5d1s2:/users/ivy
```

specifies SDF file **/users/ivy** within HP-UX device special file **/dev/rdsk/c5d1s2**

Note that this file naming convention is applicable only for use as arguments to the SDF utilities and does not constitute a legal path name for any other use within the HP-UX operating system. The shell wild-card characters *****, **?**, and **[...]** do not work for specifying an arbitrary pattern for matching SDF file names when using SDF utilities.

If the device name and a trailing colon are specified *without* a file or directory name following (for example **/dev/rdsk/c5d1s2:**), the root (**/**) of the SDF file system is assumed by convention.

Files cannot be created with the SDF utilities unless there is at least one free block of storage on the device.

Although Shared Resource Management (SRM) storage media implement the SDF file system, HP does not support the use of SDF utilities on SRM workstation storage media.

WARNINGS

SDF utilities are intended to be run on non-Series 500 HP-UX systems. If the SDF utilities are executed on a Series 500, however, *do not run them on a disk that has a mounted file system on it*.

AUTHOR

sdf(4) was developed by HP.

FILES

/tmp/SDF..LCK lock file for single user access

SEE ALSO

sdfchmod(1), sdfchown(1), sdfcp(1), sdfdf(1M), sdffind(1), sdffsck(1M), sdffsdb(1M), sdfls(1), sdfmkdir(1), sdfrm(1).

NAME

services - service name data base

DESCRIPTION

The file `/etc/services` associates official service names and aliases with the port number and protocol the services use. For each service a single line should be present with the following information:

<official service name> <port number/protocol name> <aliases>

Reserved port numbers 0 through 255 are assigned by RFC 1010.

Aliases are other names under which a service is known. Library routines such as `getservbyname()` can be invoked with a service alias instead of the service official name. For example:

```
shell    514/tcp     cmd
```

In this example, `getservbyname()` can be invoked with `cmd` instead of `shell`:

```
sp = getservbyname("cmd", "tcp");
```

instead of

```
sp = getservbyname("shell", "tcp");
```

Both produce the same results.

A line cannot start with a space or tab. Items are separated by any number of blanks (space or tab characters in any combination). The port number and protocol name are considered a single *item*. A `/` is used to separate the port and protocol (for example, `512/tcp`). A `#` character indicates the beginning of a comment. Characters from the `#` to the end of the line are not interpreted by routines which search the file.

Service names can contain any printable character other than a white space, newline, or comment character. Trailing blanks (spaces or tabs) are allowed at the end of a line.

EXAMPLES

```
shell           514/tcp    cmd
telnet          23/tcp
login           513/tcp
```

AUTHOR

`services` was developed by the University of California, Berkeley.

FILES

`/etc/services`

SEE ALSO

getservent(3N).

NAME

shells - list of allowed login shells

SYNOPSIS

`/etc/shells`

DESCRIPTION

`/etc/shells` is an ASCII file containing a list of legal shells on the system. Each shell is listed in the file by its absolute path name.

Lines or portions of lines beginning with # are assumed to be comments and are ignored. Blank lines are also ignored.

AUTHOR

`shells` was developed by HP and the University of California, Berkeley.

FILES

`/etc/shells`

SEE ALSO

chsh(1), ftpd(1M), getusershell(3C).

NAME

sm, sm.bak, state - statd directory and file structures

SYNOPSIS

`/etc/sm`

`/etc/sm.bak`

`/etc/state`

DESCRIPTION

`/etc/sm` and `/etc/sm.bak` are directories generated by `statd` (see *statd*(1M)). Each file in `/etc/sm` represents one or more machines to be monitored by the `statd` daemon. Each file in `/etc/sm.bak` represents one or more machines to be notified by the `statd` daemon upon its recovery.

`/etc/state` is a file generated by `statd` to record its version number. This version number is incremented each time a crash or recovery takes place.

SEE ALSO

statd(1M), lockd(1M).

NAME

snmpd.conf - configuration file for the SNMP agent

DESCRIPTION

When invoked, the SNMP agent reads its configuration information from the `/etc/snmpd.conf` configuration file. The SNMP agent is either the *snmpd*(1M) or the *snmpd.ea*(1M). The SNMP agent operates correctly if no values are configured in `/etc/snmpd.conf`.

`/etc/snmpd.conf` contains the following configurable values:

`get-community-name:`
: Specifies community name for the agent. The agent responds to SNMP GetRequests with this community name. You can configure the agent to respond to more than one get community name. If a community name is not entered, the agent responds to SNMP GetRequests using any community name.

`set-community-name:`
: Specifies community name for the agent. The agent responds to the SNMP SetRequests with this community name. You can configure the agent to respond to more than one set community name. If a community name is not entered, the agent returns an error.

`trap-dest:`
: Specifies the system name where traps are sent (that is, the trap destination). This system name is usually the host name or IP address of the manager.

`location:`
: Specifies the physical location of the agent.

`contact:`
: Specifies the person responsible for this agent and information on how to contact this person.

Separate the fields by blanks or tabs. A # character indicates the beginning of a comment; characters from the # character to the end of the line are ignored.

EXAMPLES

Each line in the following example `snmpd.conf` file is preceded by a comment (beginning with #) that explains the entry.

```
# Restrict the agent to responding only to
# SNMP GetRequests that have the community name "secret"
get-community-name: secret

# Allow the agent to respond to SNMP SetRequests with
# either the community name "private" or "secret"

set-community-name: private
set-community-name: secret

# Allow the agent to respond to SNMP SetRequests
# that have the community name "private"

set-community-name: private

# Send traps to system names manager1 and 15.2.113.233

trap-dest: manager1
trap-dest: 15.2.113.233

# Specify the agent is located on the first floor
# near the mens room

location: 1st Floor near Mens Room

# Specify Bob Jones is responsible for this agent
# and his phone number is 555-2000

contact: Bob Jones (Phone 555-2000)
```

AUTHOR

`snmpd.conf` was developed by HP.

SEE ALSO

chksnmpd(1), snmpd(1M), snmpd.ea(1M).

RFC 1155, RFC 1157, RFC 1212, RFC 1213

NAME

softkeys - keysh softkey file format

BACKGROUND

keysh softkey information is stored in the form of a softkey node hierarchy. The top level of this hierarchy represents the softkey commands themselves; lower levels represent various command options and parameters.

The softkey labels form a **window** into this softkey node hierarchy through which the user can view and select **eligible** nodes. A node is eligible if it was:

- Enabled by default and has not been subsequently disabled by the selection of some sibling node, or
- Disabled by default, has not been subsequently disabled by the selection of some sibling node, but has been subsequently enabled by the selection of some sibling node.

When a softkey node is selected, it can enable or disable any of its siblings as appropriate. A new window into the softkey node hierarchy is then computed as follows:

- If the selected node was not a leaf node, its eligible children are displayed;
- Otherwise, if the node still has eligible siblings remaining, they are redisplayed;
- Otherwise, if the node's parent still has eligible siblings remaining, they are redisplayed, and so on, moving up the node hierarchy.

This process of node display and selection continues until the user has entered a complete command.

At that point, **keysh** performs the **editrules** associated with each of the selected softkey nodes. These editrules create the HP-UX command that is fed to the shell for execution.

SOFTKEY FILE FORMAT

Each softkey file contains one or more softkey definitions, each of which is represented as a sub-hierarchy of **softkey nodes**.

There are two basic types of softkey nodes:

option "Options" show up on softkey labels and insert literal text into the command-line when selected. Examples are command and option names.

string "Strings" (or "parameters") show up on softkey labels but do not insert text into the command-line when selected; rather, they display a hint message. The user must then type the desired text into the command-line. Examples are file and user names.

Note that the keyword **softkey** can be used as a synonym for the keyword **option**.

The basic softkey node definition is composed of the following components:

{**option** | **string**} *softkey*
attribute
.
.
.
;

Where *softkey* is the softkey node name from which the command-line text and softkey label are derived. If necessary, a single plus sign (+) within *softkey* can be used to force hyphenation of the softkey label at a syllable boundary.

If a softkey node has an associated sub-menu, its trailing *;* is replaced with a list of child nodes as follows:

{
softkey node
.
.
.
}

Each softkey node can have the following optional *attribute* fields:

disable *count*	Selecting this node will disable *count* softkey nodes to the right of this one - default is 0.
enable *count*	Selecting this softkey will enable *count* softkey nodes to the right of this one - default is 0.
{**filter**|**command**}	This node is only active for filters or commands, respectively - default is either.
{**motorola**|**precision**}	This node is only active when **keysh** is running on a Motorola (MC680x0) or precision (PA-RISC) processor, respectively - default is either.
disabled	This node starts out disabled and must be enabled to be used - default is to start out enabled.
automatic	The command will be entered automatically when this node is selected.
editrule *editrule*	The editrule for this node.
cleanuprule *editrule*	An editrule to be executed *after* all other editrules associated with this softkey command - only one cleanuprule is allowed per softkey command.
hint *string*	The one line hint for this node - only valid for "string" softkey nodes.
help *helptext*	The help for this node (may be more than one line).
required *string*	The one-line error message to display if this node is not selected.

Arguments are as follows:

count	A signed integer, the word **none**, or the word **all**.
editrule	An editrule (described below).
helptext	**nroff**-style help enclosed in quotes (also described below).
string	An arbitrary string enclosed in quotes. Note that within quotes, \ escapes the next character as when using *awk*(1).

A typical backup softkey node definition resembles:

backup softkey *softkey* [**literal** *literal*] **;**

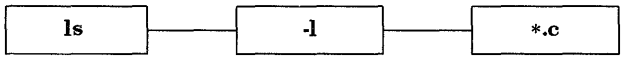
where *literal* is the literal text string to program the terminal function key with (if different than *softkey*).

An unquoted hash-mark character (#) in a softkey file delimits a comment to the end of the line.

Softkey Command Translation

To translate softkey commands into HP-UX commands, **keysh** executes the *editrules* associated with each softkey selected by the user. These editrules create a **word list** via an **awk**-like editing language. This translated word list is then passed to the shell for execution.

For a simple translation, this list might resemble:



word[0] word[1] word[2]

Every time an editrule is invoked, the special constants **last** and **next** are defined to the index of the last word in the list ("2" in this example) and the would-be-next word in the list ("3" in this example), respectively. In addition, the constant **argument** is set equal to the user input for the softkey (e.g., ***.c** for the softkey corresponding to the file name in this example).

Note that **keysh** automatically casts numbers and strings back and forth as necessary to carry out editrules. Also, variables are cleared only before the first editrule associated with a softkey command. All assigned variables are available to subsequent editrules.

Editrules

An editrule is a list of edit statements enclosed in curly-braces (i.e., `{` and `}`).

An edit statement is:

- an expression followed by a `;`,
- an `if` statement, or
- a word allocation statement.

Expressions

A simple expression can be any of:

variable	single letter from a to z
number	unsigned integer
string	enclosed in quotes
char	enclosed in quotes
last	see above
next	see above
argument	see above
motorola	boolean flag
precision	boolean flag
command	boolean flag
filter	boolean flag
word[*number*]	see above

Simple expressions can be combined with any of:

string[*number*]	single-character substring
string[*number*,*number*]	multiple-character substring
number+*number*	addition
number-*number*	subtraction
*number***number*	multiplication
number/*number*	division
number%*number*	modulus
string&*string*	concatenation
-*number*	negation
string==*string*	equality
string!=*string*	inequality
number>=*number*	greater than or equal
number<=*number*	less than or equal
number>*number*	greater than
number<*number*	less than
number&&*number*	logical and
number||*number*	logical or
!*number*	logical not
(*string*)	grouping

The following functions are also supported and return the indicated results:

`strlen`(*string*)	number of characters in *string*
`strchr`(*string*, *char*)	index of first <char> in <string>, or −1
`strrchr`(*string*, *char*)	index of last *char* in *string*, or −1
`trim`(*string*)	*string* without leading/trailing blanks
`hex`(*number*)	*number* in hex with leading `0x`
`octal`(*number*)	*number* in octal with leading `0`

Assignments can be done with any of:

variable=*string*	simple assignment
variable+=*number*	add and assign
variable-=*number*	subtract and assign
*variable**=*number*	multiply and assign
variable/=*number*	divide and assign
variable%=*number*	modulus and assign
variable&=*string*	concatenate and assign

`word[`*number*`]`=*string*	simple assignment
`word[`*number*`]`+=*number*	add and assign
`word[`*number*`]`-=*number*	subtract and assign
`word[`*number*`]`*=*number*	multiply and assign
`word[`*number*`]`/=*number*	divide and assign
`word[`*number*`]`%=*number*	modulus and assign
`word[`*number*`]`&=*string*	concatenate and assign

`if` Statement

The `if` statement is similar to the full-block mode `if` statement in awk, and is structured as follows:

```
if (number) {
    edit statement
      .
      .
      .
} else {
    edit statement
      .
      .
      .
}
```

Where the `else` part is optional. If *number* is non-zero, the first block of *edit statement*s is executed. Otherwise, if the second block of *edit statement*s is present, it is executed.

Word Allocation Statements

Word allocation statements include the following:

`insert(`*number*`,` *string*`);`	Insert *string* as a new word in the word list immediately before `word[`*number*`]`.
`append(`*string*`);`	Insert *string* as a new word in the word list immediately after the last word in the word list. Equivalent to `insert(next,` *string*`);`.
`dash(`*string*`);`	Append *string* to the last word in the word list *if* that word already begins with a dash. Otherwise, a dash is inserted as a new word in the word list immediately after the last word in the word list and *string* is appended to that.
`delete(`*number*`);`	Delete `word[`*number*`]` from the word list.

Helptext

Each softkey node can have an associated *helptext*, to be displayed upon a user request for help. This helptext is formatted on-the-fly and presented to the user through the preferred pager.

The helptext format is an nroff-like language, supporting a subset of the *man*(5) macros used to write standard HP-UX manual entries. In particular, this subset includes:

`.nf`	Begin **no-fill** mode. Display text as-is, preserving new-lines and spaces, until a `.fi`.
`.fi`	Resume **fill** mode. Display text with words filled onto each output line, attempting to utilize 90% of the screen width. (This is the default mode.)
`.br`	Force a `break` in the current output line. Display subsequent text on the next line.

`.sp`	Force a break and then display a single blank line (a vertical **space**).
`.P`	Force a break, display a single blank line, and then begin a new **paragraph** with no indent.
`.IP` *tag indent*	Force a break, display a single blank line, and then display the specified *tag*, then begin a new *indented paragraph* with the specified *indent*.
`.IL` *tag indent*	Begin a new **indented line** (similar to `.IP` except no blank line is displayed).

Note that these macros are recognized *anywhere* in the input helptext, not just at the beginning of a line. Also, all macro arguments *must be present*, even if they consist of nothing more than a quoted empty string.

EXAMPLES

For a custom `cd` command (see *cd*(1)):

```
softkey cd
editrule { append("cd"); }
{
  softkey keysh-src disable all
  editrule { append("~/keysh/src"); }
  ;
  softkey keysh-test disable all
  editrule { append("~/keysh/test"); }
  ;
  softkey keysh-doc disable all
  editrule { append("~/keysh/doc"); }
  ;
  softkey demo disable all
  editrule { append("~/demo"); }
  ;
  softkey tmp disable all
  editrule { append("/tmp"); }
  ;
  string <dir> disable all
  editrule { append(argument); }
  required "Enter the name of the directory to move to."
  ;
}
```

For other examples, refer to the file `/usr/keysh/C/softkeys`.

AUTHOR

`keysh` was developed by HP and AT&T.

FILES

`$HOME/.softkeys`	user softkey definitions file
`/usr/keysh/$LANG/softkeys`	standard softkey definitions file

SEE ALSO

keysh(1), man(5).

NAME

Switchinfo - SwitchOver/UX configuration file

DESCRIPTION

A single configuration file, **/etc/switch/Switchinfo**, contains the information required by SwitchOver/UX software. **/etc/switch/Switchinfo** is a text file, created and maintained by the System Administrator. A copy of the file resides on each host.

/etc/switch/Switchinfo contains a series of "sections", one per host. Each section begins with the name of the host, beginning in the first column, followed by a **:**. On subsequent lines are a series of "entries", one per line. Each entry is indented, and consists of a keyword, followed by =, then a value. Comments in the file begin with **#** and extend until the end of the line. White space is ignored. For example, the following is a syntactically correct (though incomplete) **Switchinfo** file:

```
# info file
hosta: # the standby
        lan0 = 0x1122334455
        prihost0 = hostb
hostb:
        standby = hosta
        lan0=0x1122334456
```

For each host, the host name should be the name that is printed by the **hostname** command when run on that host (see *hostname*(1)).

The following list names each keyword that can appear in **Switchinfo** and other pertinent information: whether the keyword is "required" or "optional"; the default value if optional; "standby" "primary", or both, depending on which sections should contain the keyword; keyword meaning; commands that use the field; and an example.

lan# Required.

Primary and standby.

Gives the new LAN station address for the network specified by **#**. The address is obtained from HP, and is not the address normally recognized by the LAN interface card. There is one address per network, and thus one entry per network.

The # denotes a single character, which is **0-9** or **a-f**. This character is the logical unit (lu) number for the LAN card, and is the same character that is used in the device file name: **lan0** names the network connected to **/dev/lan0**, and so on. (For the correspondence between logical units and slots, use the **ioscan -k** command). SwitchOver/UX requires that each LAN card connected to a particular network have the same logical unit number. So, for example, if two LANs connect Host A and Host B, and one of the LANs is **lan0** on Host A, then the same LAN must also be **lan0** on Host B. This is most easily accomplished by having all cards attached to a particular network be placed at the same hardware address on each system processor.

Supported networks include Ethernet and FDDI LAN.

Used by **switchsetlan**, **switchheartb**, and **switchreadp**.

Example:

```
lan0 = 0x1122334455
lan1 = 0x1122334456
```

standby
Required.

Primary only.

Names the host (by "hostname") that serves as standby for this primary.

Used by **switchheartb**.

Example:

```
standby = mystandbyhost
```

`rtprio`
Optional; default 0.

Primary and standby.

Gives the real-time priority of the `heartbeat` daemon on a primary or the `readpulse` daemon on a standby. The priority is a number from `0` to `127`, with 0 being highest priority; see *rtprio*(2). The value can also be `no`, meaning that the daemon should not run at a real-time priority. Real-time priority should be used if the host in question is running other real-time processes that might prevent the state-of-health monitors from completing their handshakes.

Used by `switchheartb`, and `switchreadp`.

Example:

```
rtprio = 50
```

`lockmem`
Optional; default "yes".

Primary and standby.

Tells whether the `heartbeat` daemon (primary) or the `readpulse` daemon (standby) should be locked into memory. The value can be `yes` or `no`. Locking the daemons into memory further ensures timely execution of the state-of-health daemons.

Used by `switchheartb`, and `switchreadp`,

Example:

```
lockmem = yes
```

`pulserate`
Optional; default "10".

Primary only.

Gives the interval, in seconds, between successive heartbeat messages sent from the primary to the standby over each network specified by a `lan#` entry. See `timeout` for further information.

Used by `switchheartb`.

Example:

```
pulserate = 30
```

`logfile`
Optional; default no logfile.

Primary and standby.

Names the file to which **heartbeat** (primary) and **readpulse** (standby) log errors. These daemons carefully detach themselves from any controlling terminal they may have inherited, so this entry provides the only reliable way to ensure that the daemons can send error messages to a terminal (for example, the console). If the error output should go to a file, this can be done by either using a "logfile" entry, or redirecting standard error in the place where **switchheartb** or **switchreadp** is invoked. The full pathname of **logfile** must be given.

Used by **switchheartb**, **switchreadp**.

Example:

```
logfile = /usr/adm/switch.errlog
```

prihost#
Required.

Standby only.

Names a primary host that this (standby) host will monitor. The # is a single character from **0-9**. This number is not related to the number in the **lan#** entries. However, the number is related to the **rootdisk#**, **rootmir#**, and **rootthird#** entries: **prihost0** uses **rootdisk0**, **rootmir0**, and **rootthird0**, and so on.

Used by **switchreadp** and **switchdisk1**.

Example:

```
prihost0 = myprimaryhost
```

rootdisk#
Required.

Standby only.

Names the (raw) root disk of the corresponding primary host. The # is a single character from **0-9**. **rootdisk0** corresponds to **prihost0**, and so on. It is possible, in an asymmetric SwitchOver/UX configuration, for a given disk to have different names on the primary and the standby hosts. The name here is used by the standby, and so should be the standby's name. The hardware path of this device is named by **bootpath**.

If the primary's root is mirrored using DataPair, this is the name of the primary half of the mirror (as opposed to the secondary half; note the two uses of "primary" here).

If the primary's root is a logical volume, this is the name of the physical volume created as the boot device in the root volume group.

Used by **switchreadp** and **switchdisk1**.

Example:

```
rootdisk0 = /dev/rdsk/c0d0s4
```

rootmir#
Optional; default none.

Standby only.

Similar to **rootdisk#**, except that it names the second root device. This entry should be present if the root is mirrored and absent if it is not. The hardware path of this device is named by **bootmir**.

If the primary's root is mirrored using DataPair, this is the name of the secondary half of the mirror.

If the primary's root is mirrored using Logical Volume Manager, this is the name of a second physical volume created as a boot device in the root volume group.

Used by **switchreadp** and **switchdisk1**.

Example:

```
rootmir0 = /dev/rdsk/c8d0s4
```

rootthird#
Optional; default none.

Standby only.

Similar to **rootmir#**, except that it names the third root device. This entry should be present if the root is mirrored three-way using the Logical Volume Manager, and absent if it is not. The third root device is a third physical volume created as a boot device in the root volume group. The hardware path of this device is named by **bootthird**.

Used by **switchreadp** and **switchdisk1**.

Example:

```
rootthird0 = /dev/rdsk/c6d0s2
```

timeout
Required.

Primary only.

Gives the heartbeat timeout value, in seconds. If **readpulse** has not seen a heartbeat message from the primary after this period of time, then it runs **become** to start the takeover. In general, **timeout** should be larger than any **pulserates** so that failover occurs only when a primary has truly failed. As an example, **timeouts** slightly larger than twice the largest **pulserate** would avoid failover in the event of losing one readpulse message on the network.

Used by **switchreadp**.

Example:

```
timeout = 60
```

bootpath
Required.

Primary only.

Gives the hardware address of the primary boot device, which the standby system processor should use when it reboots the primary host. The standby changes its bootpath in stable storage to this value before it reboots. The device located at this hardware address is named by **rootdisk**. For mirrored root devices, see the description of **bootmir** and **bootthird** below.

Note that, although this value appears in a file organized by host, the value is in fact dependent on the system processor, and not the host. This is not an issue with SwitchOver/UX, because of the requirement that each disk be accessed through identical hardware paths by any system processor that can access it.

Used by **switchreadp**.

Example:

```
bootpath = 4.4.0
```

bootmir
Optional; default none.

Primary only.

Gives the hardware address of the alternate boot device. If present, the standby host checks the primary boot device, and if it is unavailable, sets its primary bootpath to **bootmir** before rebooting. The device at this hardware address is named by **rootmir.**

For mirroring with DataPair, the alternate boot device is the secondary half of the mirrored root device.

For mirroring with the Logical Volume Manager, the alternate boot device is a second physical volume created as a boot device.

Used by **switchreadp**.

Example:

```
bootmir = 4.3.0
```

bootthird
Optional; default none.

Primary only.

Gives the hardware address of the third boot device. This entry is required if root is three-way mirrored using the Logical Volume Manager, and absent if it is not. The third boot device is a third physical volume in the root volume group created as a boot device. If present, the standby host checks the other two boot devices and if both are unavailable, sets its primary boot address to **bootthird** before rebooting. The device at this hardware address is named by **rootthird**.

Used by **switchreadp**.

Example:

```
bootmir = 4.3.0
```

rootdisk
Required.

Primary and standby.

Gives the name of the (raw) root disk for the given host. In an asymmetric SwitchOver/UX configuration, a given disk can have different names on the primary and standby hosts; here, the name is the name used by the host in whose section this entry appears. (Note the difference between this entry and the **rootdisk#** entry, which appears only in the standby section, and which gives the standby name for the disk.) The hardware path of this device is named by **bootpath**.

If the disk is mirrored using DataPair, this is the name of the primary half of the mirror, and **rootmir** is the name of the secondary.

If root is a logical volume, this names the physical volume created as the primary boot device in the root volume group.

Used by **switchdisk1**.

Example:

```
rootdisk = /dev/rdsk/c0d0s4
```

rootmir
Optional.

Primary and standby.

Similar to **rootdisk**, except that it names the second root device. This entry is required if the root is mirrored and omitted if it is not. The hardware path of this device is named by **bootmir**.

If the primary's root is mirrored using DataPair, this is the name of the secondary half of the mirror.

If the primary's root is mirrored using Logical Volume Manager, this is the name of a second physical volume created as a boot device in the root volume group.

Used by **switchdisk1**.

Example:

```
rootmir = /dev/rdsk/c8d0s4
```

rootthird
Optional.

Primary and standby.

Like **rootdisk**, but gives the name of the third boot device. This entry is required if the root is mirrored three-way using the Logical Volume Manager, and should be omitted if it is not. This device is a third physical volume in the root volume group created as a boot device. The hardware path of this device is named by **bootthird**.

Used by **switchdisk1**.

Example:

```
rootthird = /dev/rdsk/c8d0s4
```

SEE ALSO
switchdiskl(1M), switchheartb(1M), switchreadp(1M), switchsetflg(1M), switchsetlan(1M).

NAME

symlink - symbolic link

DESCRIPTION

A symbolic link is a type of file that indirectly refers to ("points to") a path name. Also known as a **soft link**, a symbolic link contains a relative or absolute path name. If a symbolic link to a relative path name is encountered during path name interpretation, the contents of the symbolic link replaces the symbolic link component and are expanded into the path name being interpreted. If a symbolic link to an absolute path name is encountered, the contents of the symbolic link replaces all components up to and including the symbolic link, and is expanded into the remainder of the path name.

Thus, given path name `/a/b/c/d`, where `c` is a symbolic link to `../x/y`, the original path name is interpreted as `/a/b/../x/y/d`. If, instead, `c` is a symbolic link to an absolute path name such as `/v/w`, the same path name would be interpreted as `/v/w/d`. All symbolic links are interpreted in this manner except when the symbolic link is the last component of a path name passed as a parameter to one of the system calls: `readlink`, `rename`, `symlink`, `unlink`, `chown`, or `lstat` (see *readlink*(2), *rename*(2), *symlink*(2), *unlink*(2), *chown*(2) and *lstat*(2)). With these calls, the symbolic link itself is accessed or affected.

Unlike normal (hard) links, a symbolic link can refer to any arbitrary path name and can span different logical devices (volumes). The path name can be that of any type of file (including a directory or another symbolic link), or it can even be invalid if no such path exists in the system. Thus it is possible to make symbolic links point to themselves or other symbolic links in such a way that they form a closed loop. The system detects this situation by limiting the number of symbolic links it traverses while translating a path name. The mode and ownership of a symbolic link is ignored by the system, which means that `chmod` affects the actual file; not the file containing the symbolic link (see *chmod*(1)).

Symbolic links can be created using `ln` or `symlink` (see *ln*(1) and *symlink*(2)).

AUTHOR

`symlink` was developed by HP and the University of California, Berkeley.

SEE ALSO

cp(1), symlink(2), readlink(2), link(2), stat(2), mknod(1M).

NAME

tar - format of tar tape archive

DESCRIPTION

The *header* structure produced by `tar` (see *tar*(1)) is as follows (the array size defined by the constants is shown on the right):

```
struct {
    char name[NAMSIZ];          (100)
    char mode[MODE_SZ];           (8)
    char uid[UID_SZ];             (8)
    char gid[GID_SZ];             (8)
    char size[SIZE_SZ];          (12)
    char mtime[MTIME_SZ];        (12)
    char chksum[CHKSUM_SZ];       (8)
    char typeflag;
    char linkname[NAMSIZ];      (100)
    char magic[MAGIC_SZ];         (6)
    char version[VERSION_SZ];     (2)
    char uname[UNAME_SZ];        (32)
    char gname[GNAME_SZ];        (32)
    char devmajor[DEV_SZ];        (8)
    char devminor[DEV_SZ];        (8)
    char prefix[PREFIX_SZ];     (155)
} dbuf;
```

All characters are represented in ASCII. There is no padding used in the header block; all fields are contiguous.

The fields *magic*, *uname*, and *gname* are null-terminated character strings. The fields *name*, *linkname*, and *prefix* are null-terminated character strings except when all characters in the array contain non-null characters, including the last character. The *version* field is two bytes containing the characters `00` (zero-zero). The *typeflag* contains a single character. All other fields are leading-zero-filled octal numbers in ASCII. Each numeric field is terminated by one or more space or null characters.

The *name* and the *prefix* fields produce the pathname of the file. The hierarchical relationship of the file is retained by specifying the pathname as a path prefix, with a slash character and filename as the suffix. If the *prefix* contains non-null characters, *prefix,* a slash character, and *name* are concatenated without modification or addition of new characters to produce a new pathname. In this manner, pathnames of at most 256 characters can be supported. If a pathname does not fit in the space provided, the format-creating utility notifies the user of the error, and no attempt is made to store any part of the file, header, or data on the medium.

In the HP Clustered environment, CDFs are located by appending a `+` onto *name* and testing for a directory using `stat()` (see *stat*(2)). If the test fails, the `+` is removed.

SEE ALSO

tar(1), tar(5)

STANDARDS CONFORMANCE

`tar`: XPG4, FIPS 151-2, POSIX.1

NAME

term - format of compiled term file

SYNOPSIS

`term`

DESCRIPTION

Compiled terminfo descriptions are placed under the directory `/usr/lib/terminfo`. In order to avoid a linear search of a huge HP-UX system directory, a two-level scheme is used: `/usr/lib/terminfo/`*c*`/`*name* where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, `hp110` can be found in the file `/usr/lib/terminfo/h/hp110`. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it is the same on all hardware. An 8-bit or longer byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created using the `tic` program (see *tic*(1M)), and read by the `setupterm()` routine. Both of these pieces of software are part of the *curses*(3X) package. The file is divided into the following six parts:

header This section begins the file and contains six short integers in the following format:

1. Magic number (octal 0432);
2. Size, in bytes, of the names section;
3. Number of bytes in the Boolean section;
4. Number of short integers in the numbers section;
5. Number of offsets (short integers) in the strings section;
6. Size, in bytes, of the string table.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value; the second byte contains the most significant 8 bits. (Thus, the value represented is 256 * second + first.) The value −1 is represented by `0377, 0377`; other negative values are illegal. The −1 generally means that a capability is missing from this terminal. Note that this format corresponds to the hardware of the VAX and PDP-11. Machines where this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the `|` character. The section is terminated with an ASCII NUL character.

The Boolean flags have one byte for each flag. This byte is either `0` or `1` as the flag is absent or present, respectively. The capabilities are in the same order as they are listed in the file `<term.h>`.

Between the Boolean section and the number section, a null byte will be inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short word boundary.

The numbers section is similar to the flags section. Each capability consists of two bytes, and is stored as a short integer. If the value represented is -1, the capability is considered missing.

The strings section is also similar. Each capability is stored as a short integer in the format above. A value of −1 means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^*X* or *c* notation are stored in their interpreted form, not the printing representation. Padding information $*nn* and parameter information %*x* are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null terminated.

Note that it is possible for `setupterm()` to expect a different set of capabilities than are actually present in the file. Either the database might have been updated since `setupterm()` has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine `setupterm()` must be prepared for both possibilities, which is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of Boolean, number, and string capabilities.

The following example is an octal dump of the description for the HP Portable Computer (HP-110):

```
110|hp110|hp110a portable computer,
    am, xhp, da, db, mir, cols#80, lines#16, lm#0,
    cbt=\E1, bel=^G, cr=\r, tbc=\E3, clear=\E&a0y0C\EJ,
    el=\EK, ed=\EJ, hpa=\E&a%p1%dC, cup=\E&a%p1%dy%p2%dC,
    cud1=\EB, cub1=\b, cuf1=\EC, cuu1=\EA, cvvis=\E&j@,
    dch1=\EP, dl1=\EM, smir=\EQ, smso=\E&dB, sgr0=\E&d@,
    rmir=\ER, rmso=\E&d@, is2=\E&j@,
    if=/usr/lib/tabset/stdcrt, ill=\EL, kbs=\b, kcud1=\EB,
    khome=\Eh, kcub1=\ED, kcuf1=\EC, kcuu1=\EA, rmkx=\E&s0A,
    smkx=\E&s1A, vpa=\E&a%p1%dY, ind=\n, hts=\E1, ht=\t,

0000 032 001   #  \0 025  \0  \b  \0 223  \0 254  \0   1   1   0   |
0020   h   p   1   1   0   |   h   p   1   1   0   a       p   o   r
0040   t   a   b   l   e       c   o   m   p   u   t   e   r  \0  \0
0060 001  \0 001  \0  \0  \0  \0  \0  \0  \0 001 001 001  \0  \0  \0
0100  \0  \0  \0  \0   P  \0 377 377 020  \0  \0  \0 377 377 377 377
0120 377 377 377 377  \0  \0 003  \0 005  \0 377 377 007  \0  \n  \0
0140 024  \0 027  \0 032  \0 377 377   $  \0   4  \0 377 377 377 377
0160   7  \0 377 377 377 377   9  \0 377 377   <  \0   ?  \0   D  \0
0200   G  \0 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0220 377 377   J  \0 377 377 377 377 377 377   M  \0 377 377 377 377
0240 377 377   R  \0 377 377 377 377   W  \0   Z  \0 377 377 377 377
0260 377 377 377 377 377 377   _  \0 377 377   d  \0 377 377   {  \0
0300 377 377   ~  \0 377 377 377 377 377 377 377 377 377 377 200  \0
0320 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0340 377 377 377 377 377 377 377 377 377 377 377 377 203  \0 377 377
0360 377 377 206  \0 377 377 377 377 377 377 211  \0 377 377 377 377
0400 377 377 214  \0 217  \0 225  \0 377 377 377 377 377 377 377 377
0420 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377

0520 377 377 233  \0 377 377 245  \0 377 377 377 377 247  \0 377 377
0540 252  \0 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0560 377 377 377 377 377 377 377 377 377 377 033   i  \0 007  \0  \r
0600  \0 033   3  \0 033   &   a   0   y   0   C 033   J  \0 033   K
0620  \0 033   J  \0 033   &   a   %   p   1   %   d   C  \0 033   &
0640   a   %   p   1   %   d   y   %   p   2   %   d   C  \0 033   B
0660  \0  \b  \0 033   C  \0 033   A  \0 033   &   j   @ \0 033   P
0700  \0 033   M  \0 033   Q  \0 033   &   d   B  \0  033  &   d  @
0720  \0 033   R  \0 033   &   d   @ \0 033   &   j   @ \0   /   u
0740   s   r   /   l   i   b   /   t   a   b   s   e   t   /   s   t
0760   d   c   r   t  \0 033   L  \0  \b  \0 033   B  \0 033   h  \0
1000 033   D  \0 033   C  \0 033   A  \0 033   &   s   0   A  \0 033
1020   &   s   1   A  \0 033   &   a   %   p   1   %   d   Y  \0  \n
1040  \0 033   1  \0  \t  \0
1046
```

WARNINGS

Total compiled entries cannot exceed 4096 bytes.

The name field cannot exceed 128 bytes.

Hewlett-Packard Company supports only those terminals that are listed on the current list of supported devices. However, both non-supported and supported terminals may be in the terminfo database. If non-supported terminals are used, they may not work correctly.

FILES

`/usr/lib/terminfo/?/*` compiled terminal capability data base

SEE ALSO

tic(1M), untic(1M), curses(3X), terminfo(4).

NAME

terminfo - terminal capability database

SYNOPSIS

`/usr/lib/terminfo/?/*`

DESCRIPTION

`terminfo` is a database describing terminals used by, for example, *vi*(1) and *curses*(3X). Terminals are described in `terminfo` by giving a set of capabilities which they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in `terminfo`.

Entries in `terminfo` consist of a number of comma (`,`) separated fields. White space after each comma is ignored. The first entry for each terminal gives the names which are known for the terminal, separated by vertical bar (`|`) characters. The first name given is the most common abbreviation for the terminal, the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should be in lowercase and contain no blanks; the last name can contain uppercase and blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, thus "hp2621". This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. Thus, a vt100 in 132 column mode would be vt100-w. The following suffixes should be used where possible:

Suffix	Meaning	Example
`-w`	Wide mode (more than 80 columns)	`vt100-w`
`-am`	With auto. margins (usually default)	`vt100-am`
`-nam`	Without automatic margins	`vt100-nam`
-n	Number of lines on the screen	`aaa-60`
`-na`	No arrow keys (leave them in local)	`c100-na`
-n`p`	Number of pages of memory	`c100-4p`
`-rv`	Reverse video	`c100-rv`

Capabilities

The variable is the name by which the programmer (at the terminfo level) accesses the capability. The *cap-name* is the short name used in the text of the database, and is used by a person updating the database. The *i.code* is the two letter internal code used in the compiled database, and always corresponds to the old `termcap` capability name.

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short and to allow the tabs in the source file `caps` to line up nicely. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification. In the table below:

(P) indicates that padding may be specified

(G) indicates that the string is passed through tparm with parms as given (#*i*).

(*) indicates that padding may be based on the number of lines affected

($\#_i$) indicates the *i*th parameter.

Variable Booleans	Cap-name	I. Code	Description
`auto_left_margin,`	`bw`	`bw`	cubl wraps from column 0 to last column
`auto_right_margin,`	`am`	`am`	Terminal has automatic margins
`beehive_glitch,`	`xsb`	`xb`	Beehive (f1=escape, f2=ctrl C)
`ceol_standout_glitch,`	`xhp`	`xs`	Standout not erased by overwriting (hp)
`eat_newline_glitch,`	`xenl`	`xn`	newline ignored after 80 cols (Concept)
`erase_overstrike,`	`eo`	`eo`	Can erase overstrikes with a blank
`generic_type,`	`gn`	`gn`	Generic line type (e.g.,, dialup, switch).
`hard_copy,`	`hc`	`hc`	Hardcopy terminal
`has_meta_key,`	`km`	`km`	Has a meta key (shift, sets parity bit)
`has_status_line,`	`hs`	`hs`	Has extra "status line"
`insert_null_glitch,`	`in`	`in`	Insert mode distinguishes nulls

memory_above,	da	da	Display may be retained above the screen
memory_below,	db	db	Display may be retained below the screen
move_insert_mode,	mir	mi	Safe to move while in insert mode
move_standout_mode,	msgr	ms	Safe to move in standout modes
over_strike,	os	os	Terminal overstrikes
status_line_esc_ok,	eslok	es	Escape can be used on the status line
teleray_glitch,	xt	xt	Tabs ruin, magic so char (Teleray 1061)
tilde_glitch,	hz	hz	Hazeltine; cannot print ~
transparent_underline,	ul	ul	underline character overstrikes
xon_xoff,	xon	xo	Terminal uses xon/xoff handshaking

Numbers:

columns,	cols	co	Number of columns in a line
init_tabs,	it	it	Tabs initially every # spaces
lines,	lines	li	Number of lines on screen or page
lines_of_memory,	lm	lm	Lines of memory if > lines. 0 means varies
magic_cookie_glitch,	xmc	sg	Number of blank chars left by smso or rmso
padding_baud_rate,	pb	pb	Lowest baud where cr/nl padding is needed
virtual_terminal,	vt	vt	Virtual terminal number (HP-UX system)
width_status_line,	wsl	ws	No. columns in status line
num_labels,	nlab	Nl	Number of labels on screen (start at 1)
label_height,	lh	lh	Number of rows in each label
label_width,	lw	lw	Number of cols in each label

Strings:

back_tab,	cbt	bt	Back tab (P)
bell,	bel	bl	Audible signal (bell) (P)
carriage_return,	cr	cr	Carriage return (P*)
change_scroll_region,	csr	cs	change to lines #1 through #2 (vt100) (PG)
clear_all_tabs,	tbc	ct	Clear all tab stops (P)
clear_screen,	clear	cl	Clear screen and home cursor (P*)
clr_eol,	el	ce	Clear to end of line (P)
clr_eos,	ed	cd	Clear to end of display (P*)
column_address,	hpa	ch	Set cursor column (PG)
command_character,	cmdch	CC	Term. settable cmd char in prototype
cursor_address,	cup	cm	Screen rel. cursor motion row #1 col #2 (PG)
cursor_down,	cud1	do	Down one line
cursor_home,	home	ho	Home cursor (if no cup)
cursor_invisible,	civis	vi	Make cursor invisible
cursor_left,	cub1	le	Move cursor left one space
cursor_mem_address,	mrcup	CM	Memory relative cursor addressing
cursor_normal,	cnorm	ve	Make cursor appear normal (undo vs/vi)
cursor_right,	cuf1	nd	Non-destructive space (cursor right)
cursor_to_ll,	ll	ll	Last line, first column (if no cup)
cursor_up,	cuu1	up	Upline (cursor up)
cursor_visible,	cvvis	vs	Make cursor very visible
delete_character,	dch1	dc	Delete character (P*)
delete_line,	dl1	dl	Delete line (P*)
dis_status_line,	dsl	ds	Disable status line
down_half_line,	hd	hd	Half-line down (forward 1/2 linefeed)
enter_alt_charset_mode,	smacs	as	Start alternate character set (P)
enter_blink_mode,	blink	mb	Turn on blinking
enter_bold_mode,	bold	md	Turn on bold (extra bright) mode
enter_ca_mode,	smcup	ti	String to begin programs that use cup
enter_delete_mode,	smdc	dm	Delete mode (enter)
enter_dim_mode,	dim	mh	Turn on half-bright mode
enter_insert_mode,	smir	im	Insert mode (enter);
enter_protected_mode,	prot	mp	Turn on protected mode
enter_reverse_mode,	rev	mr	Turn on reverse video mode

enter_secure_mode,	invis	mk	Turn on blank mode (chars invisible)
enter_standout_mode,	smso	so	Begin stand out mode
enter_underline_mode,	smul	us	Start underscore mode
erase_chars,	ech	ec	Erase #1 characters (PG)
exit_alt_charset_mode,	rmacs	ae	End alternate character set (P)
exit_attribute_mode,	sgr0	me	Turn off all attributes
exit_ca_mode,	rmcup	te	String to end programs that use cup
exit_delete_mode,	rmdc	ed	End delete mode
exit_insert_mode,	rmir	ei	End insert mode
exit_standout_mode,	rmso	se	End stand out mode
exit_underline_mode,	rmul	ue	End underscore mode
flash_screen,	flash	vb	Visible bell (may not move cursor)
form_feed,	ff	ff	Hardcopy terminal page eject (P*)
from_status_line,	fsl	fs	Return from status line
init_1string,	is1	i1	Terminal initialization string
init_2string,	is2	i2	Terminal initialization string
init_3string,	is3	i3	Terminal initialization string
init_file,	if	if	Name of file containing is
insert_character,	ich1	ic	Insert character (P)
insert_line,	il1	al	Add new blank line (P*)
insert_padding,	ip	ip	Insert pad after character inserted (p*)
key_backspace,	kbs	kb	Sent by backspace key
key_catab,	ktbc	ka	Sent by clear-all-tabs key
key_clear,	kclr	kC	Sent by clear screen or erase key
key_ctab,	kctab	kt	Sent by clear-tab key
key_dc,	kdch1	kD	Sent by delete character key
key_dl,	kdl1	kL	Sent by delete line key
key_down,	kcud1	kd	Sent by terminal down arrow key
key_eic,	krmir	kM	Sent by rmir or smir in insert mode
key_eol,	kel	kE	Sent by clear-to-end-of-line key
key_eos,	ked	kS	Sent by clear-to-end-of-screen key
key_f0,	kf0	k0	Sent by function key f0
key_f1,	kf1	k1	Sent by function key f1
key_f2,	kf2	k2	Sent by function key f2
key_f3,	kf3	k3	Sent by function key f3
key_f4,	kf4	k4	Sent by function key f4
key_f5,	kf5	k5	Sent by function key f5
key_f6,	kf6	k6	Sent by function key f6
key_f7,	kf7	k7	Sent by function key f7
key_f8,	kf8	k8	Sent by function key f8
key_f9,	kf9	k9	Sent by function key f9
key_f10,	kf10	ka	Sent by function key f10
key_f11,	kf11	F1	Sent by function key f11
key_f12,	kf12	F2	Sent by function key f12
key_f13,	kf13	F3	Sent by function key f13
key_f14,	kf14	F4	Sent by function key f14
key_f15,	kf15	F5	Sent by function key f15
key_f16,	kf16	F6	Sent by function key f16
key_f17,	kf17	F7	Sent by function key f17
key_f18,	kf18	F8	Sent by function key f18
key_f19,	kf19	F9	Sent by function key f19
key_f20,	kf20	FA	Sent by function key f20
key_f21,	kf21	FB	Sent by function key f21
key_f22,	kf22	FC	Sent by function key f22
key_f23,	kf23	FD	Sent by function key f23
key_f24,	kf24	FE	Sent by function key f24
key_f25,	kf25	FF	Sent by function key f25
key_f26,	kf26	FG	Sent by function key f26
key_f27,	kf27	FH	Sent by function key f27

key_f28,	kf28	FI	Sent by function key f28
key_f29,	kf29	FJ	Sent by function key f29
key_f30,	kf30	FK	Sent by function key f30
key_f31,	kf31	FL	Sent by function key f31
key_f32,	kf32	FM	Sent by function key f32
key_f33,	kf33	FN	Sent by function key f33
key_f34,	kf34	FO	Sent by function key f34
key_f35,	kf35	FP	Sent by function key f35
key_f36,	kf36	FQ	Sent by function key f36
key_f37,	kf37	FR	Sent by function key f37
key_f38,	kf38	FS	Sent by function key f38
key_f39,	kf39	FT	Sent by function key f39
key_f40,	kf40	FU	Sent by function key f40
key_f41,	kf41	FV	Sent by function key f41
key_f42,	kf42	FW	Sent by function key f42
key_f43,	kf43	FX	Sent by function key f43
key_f44,	kf44	FY	Sent by function key f44
key_f45,	kf45	FZ	Sent by function key f45
key_f46,	kf46	Fa	Sent by function key f46
key_f47,	kf47	Fb	Sent by function key f47
key_f48,	kf48	Fc	Sent by function key f48
key_f49,	kf49	Fd	Sent by function key f49
key_f50,	kf50	Fe	Sent by function key f50
key_f51,	kf51	Ff	Sent by function key f51
key_f52,	kf52	Fg	Sent by function key f52
key_f53,	kf53	Fh	Sent by function key f53
key_f54,	kf54	Fi	Sent by function key f54
key_f55,	kf55	Fj	Sent by function key f55
key_f56,	kf56	Fk	Sent by function key f56
key_f57,	kf57	Fl	Sent by function key f57
key_f58,	kf58	Fm	Sent by function key f58
key_f59,	kf59	Fn	Sent by function key f59
key_f60,	kf60	Fo	Sent by function key f60
key_f61,	kf61	Fp	Sent by function key f61
key_f62,	kf62	Fq	Sent by function key f62
key_f63,	kf63	Fr	Sent by function key f63
key_home,	khome	kh	Sent by home key
key_ic,	kich1	kI	Sent by ins char/enter ins mode key
key_il,	kil1	kA	Sent by insert line
key_left,	kcub1	kl	Sent by terminal left arrow key
key_ll,	kll	kH	Sent by home-down key
key_npage,	knp	kN	Sent by next-page key
key_ppage,	kpp	kP	Sent by previous-page key
key_right,	kcuf1	kr	Sent by terminal right arrow key
key_sf,	kind	kF	Sent by scroll-forward/down key
key_sr,	kri	kR	Sent by scroll-backward/up key
key_stab,	khts	kT	Sent by set-tab key
key_up,	kcuu1	ku	Sent by terminal up arrow key
keypad_local,	rmkx	ke	Out of "keypad transmit" mode
keypad_xmit,	smkx	ks	Put terminal in "keypad transmit" mode
lab_f0,	lf0	10	Labels on function key f0 if not f0
lab_f1,	lf1	11	Labels on function key f1 if not f1
lab_f10,	lf10	la	Labels on function key f10 if not f10
lab_f2,	lf2	12	Labels on function key f2 if not f2
lab_f3,	lf3	13	Labels on function key f3 if not f3
lab_f4,	lf4	14	Labels on function key f4 if not f4
lab_f5,	lf5	15	Labels on function key f5 if not f5
lab_f6,	lf6	16	Labels on function key f6 if not f6
lab_f7,	lf7	17	Labels on function key f7 if not f7

lab_f8,	lf8	l8	Labels on function key f8 if not f8
lab_f9,	lf9	l9	Labels on function key f9 if not f9
label_off,	rmln	LF	Turn off soft labels
label_on,	smln	LO	Turn on soft labels
memory_lock,	meml	ml	Lock memory above cursor
memory_unlock,	memu	mu	Turn memory lock off
meta_on,	smm	mm	Turn on "meta mode" (8th bit)
meta_off,	rmm	mo	Turn off "meta mode"
newline,	nel	nw	Newline (behaves like cr followed by lf)
pad_char,	pad	pc	Pad character (rather than null)
parm_dch,	dch	DC	Delete #1 chars (PG*)
parm_delete_line,	dl	DL	Delete #1 lines (PG*)
parm_down_cursor,	cud	DO	Move cursor down #1 lines (PG*)
parm_ich,	ich	IC	Insert #1 blank chars (PG*)
parm_index,	indn	SF	Scroll forward #1 lines (PG)
parm_insert_line,	il	AL	Add #1 new blank lines (PG*)
parm_left_cursor,	cub	LE	Move cursor left #1 spaces (PG)
parm_right_cursor,	cuf	RI	Move cursor right #1 spaces (PG*)
parm_rindex,	rin	SR	Scroll backward #1 lines (PG)
parm_up_cursor,	cuu	UP	Move cursor up #1 lines (PG*)
pkey_key,	pfkey	pk	Prog funct key #1 to type string #2
pkey_local,	pfloc	pl	Prog funct key #1 to execute string #2
pkey_xmit,	pfx	px	Prog funct key #1 to xmit string #2
plab_norm,	pln	pn	Prog label #1 to show string #2
print_screen,	mc0	ps	Print contents of the screen
prtr_off,	mc4	pf	Turn off the printer
prtr_on,	mc5	po	Turn on the printer
repeat_char,	rep	rp	Repeat char #1 #2 times. (PG*)
reset_1string,	rs1	r1	Reset terminal completely to sane modes.
reset_2string,	rs2	r2	Reset terminal completely to sane modes.
reset_3string,	rs3	r3	Reset terminal completely to sane modes.
reset_file,	rf	rf	Name of file containing reset string
restore_cursor,	rc	rc	Restore cursor to position of last sc
row_address,	vpa	cv	Vertical position absolute (set row) (PG)
save_cursor,	sc	sc	Save cursor position (P)
scroll_forward,	ind	sf	Scroll text up (P)
scroll_reverse,	ri	sr	Scroll text down (P)
set_attributes,	sgr	sa	Define the video attributes (PG9)
set_tab,	hts	st	Set a tab in all rows, current column
set_window,	wind	wi	Current window is lines #1-#2 cols #3-#4
tab,	ht	ta	Tab to next 8 space hardware tab stop
to_status_line,	tsl	ts	Go to status line, column #1
underline_char,	uc	uc	Underscore one char and move past it
up_half_line,	hu	hu	Half-line up (reverse 1/2 linefeed)
init_prog,	iprog	iP	Path name of program for init
key_a1,	ka1	K1	Upper left of keypad
key_a3,	ka3	K3	Upper right of keypad
key_b2,	kb2	K2	Center of keypad
key_c1,	kc1	K4	Lower left of keypad
key_c3,	kc3	K5	Lower right of keypad
prtr_non,	mc5p	pO	Turn on the printer for #1 bytes

A Sample Entry

The following entry, which describes the Concept-100, is among the more complex entries in the `terminfo` file as of this writing.

```
concept100|c100|concept|c104|c100-4p|concept 100,
    am, bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>, cnorm=\Ew,
    cols#80, cr=^M$<9>, cub1=^H, cud1=^J, cuf1=\E=,
    cup=\Ea%p1%' '%+%c%p2%' '%+%c,
```

```
cuu1=\E;, cvvis=\EW, db, dch1=\E^A$<16*>, dim=\EE, dl1=\E^B$<3*>,
ed=\E^C$<16*>, el=\E^U$<16>, eo, flash=\Ek$<20>\EK, ht=\t$<8>,
il1=\E^R$<3*>, in, ind=^J, .ind=^J$<9>, ip=$<16*>,
is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E,
kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;,
kf1=\E5, kf2=\E6, kf3=\E7, khome=\E?,
lines#24, mir, pb#9600, prot=\EI, rep=\Er%p1%c%p2%' '%+%c$<.2*>,
rev=\ED, rmcup=\Ev    $<6>\Ep\r\n, rmir=\E\200, rmkx=\Ex,
rmso=\Ed\Ee, rmul=\Eg, rmul=\Eg, sgr0=\EN\200,
smcup=\EU\Ev  8p\Ep\r, smir=\E^P, smkx=\EX, smso=\EE\ED,
smul=\EG, tabs, ul, vt#8, xenl,
```

Entries can continue onto multiple lines by placing white space at the beginning of each line except the first. Comments can be included on lines beginning with #. Capabilities in `terminfo` are of three types: Boolean capabilities indicating that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities that identify a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the fact that the Concept has `automatic margins` (i.e., an automatic return and linefeed when the end of a line is reached) is indicated by the capability `am`. Hence the description of the Concept includes `am`. Numeric capabilities are followed by the character `#` and then the value. Thus `cols`, which indicates the number of columns the terminal has, gives the value `80` for the Concept.

Finally, string valued capabilities, such as `el` (clear-to-end-of-line sequence) are given by the two-character code, an =, and then a string ending at the next following `,`. A delay, in milliseconds, can appear anywhere in such a capability, enclosed in `$<..>` brackets, as in `el=\EK$<3>`, and padding characters are supplied by `tputs` to provide this delay. The delay can be either a number, (such as `20`) or a number followed by `*` (as in `3*`). A `*` indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of *lines* affected. This is always `1` unless the terminal has `xenl` and the software uses it.) When a `*` is specified, it is sometimes useful to give a delay of the form `3.5` to specify a delay per unit to tenths of milliseconds (only one digit is allowed to the right of the decimal).

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. Both `\E` and `\e` map to an ESCAPE character, `^`*x* maps to Ctrl-*x* for any appropriate *x*, and the sequences `\n`, `\l`, `\r`, `\t`, `\b`, `\f`, and `\s` give a newline, linefeed, return, tab, backspace, formfeed, and space. Other escapes include `\^` for `^`, `\\` for `\`, `\,` for comma, `\:` for `:`, and `\0` for null. (`\0` produces `\200`, which does not terminate a string, but behaves as a null character on most terminals.) Finally, characters may be specified as three octal digits after a `\`.

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second `ind` in the example above.

Preparing Descriptions

This section outlines how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in `terminfo` and building up a description gradually by using partial descriptions with `vi` to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the `terminfo` file to describe it, or it may expose bugs in `vi`. To test a new terminal description easily, set the environment variable `TERMINFO` to a pathname of a directory containing the compiled description you are working on so programs look there rather than in `/usr/lib/terminfo`. To get the padding for insert line right (if the terminal manufacturer did not document it) a severe test is to edit `/etc/passwd` at 9600 baud, delete 16 or so lines from the middle of the screen, then rapidly press the `u` (undo) key several times in succession. If the terminal behaves strangely, more padding is usually needed. A similar test can be used for insert character.

Basic Capabilities

The number of columns on each line for the terminal is given by the `cols` numeric capability. If the terminal is a CRT, the number of lines on the screen is given by the `lines` capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, it should have the `am` capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the

clear string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as TEKTRONIX 4010 series, as well as hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr** (normally carriage return (Ctrl-M)). If there is a code to produce an audible signal (bell, beep, etc) give this as **bel**.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over. For example, you would not normally use "**cuf1=** " because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in **terminfo** are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge unless **bw** is given, and never attempt to go up locally off the top. In order to scroll text up, a program should move to the bottom left corner of the screen, then send the **ind** (index) string.

To scroll text down, a program moves to the top left corner of the screen and sends the **ri** (reverse index) string. Strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin** which have the same semantics as **ind** and **ri** except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion which is defined from the left edge is if **bw** is given, then a **cub1** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch selectable automatic margins, the **terminfo** file usually assumes that this is on; i.e., **am**. If the terminal has a command which moves to the first column of the next line, that command can be given as **nel** (new-line). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf**, it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and CRT terminals. Thus the model 33 teletype is described as

```
33|tty33|tty|model 33 teletype,
bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,
```

while the Lear Siegler ADM-3 is described as

```
adm3|3|lsi adm3,
am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H, cud1=^J,
ind=^J, lines#24,
```

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with *printf*(3S)-like escapes **%***x* in it. For example, to address the cursor, the **cup** capability is given by using two parameters which specify the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory-relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special **%** codes to manipulate it. Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary.

% encodings have the following meanings:

%%	outputs **%**
%d	print **pop()** as in printf
%2d	print **pop()** like **%2d**
%3d	print **pop()** like **%3d**
%02d	
%03d	as in C printf()

%c	print pop() gives %c
%s	print pop() gives %s
%p[1-9]	push ith parm
%P[a-z]	set variable [a-z] to pop()
%g[a-z]	get variable [a-z] and push it
%'c'	char constant c
%{nn}	integer constant nn
%l	push strlen(pop())
%+ %- %* %/ %m	arithmetic (%m is mod): push(pop() op pop())
%& %| %^	bit operations: push(pop() op pop())
%= %> %<	logical operations: push(pop() op pop())
%! %~	unary operations push(op pop())
%i	add 1 to first two parms (for ANSI terminals)

%? *expr* %t *thenpart* %e *elsepart* %;
if-then-else, %e *elsepart* is optional.
else-if's are possible as in Algol 68:
%? c_1 %t b_1 %e c_2 %t b_2 %e c_3 %t b_3 %e c_4 %t b_4 %e %;
c_i are conditions, b_i are bodies.

Binary operations are in postfix form with the operands in the usual order. That is, to get x-5, use %gx%{5}%-.

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its cup capability is cup=6\E&%p2%2dc%p1%2dY.

The Microterm ACT-IV needs the current row and column sent, preceded by a ^T, with the row and column simply encoded in binary, cup=^T%p1%c%p2%c. Terminals that use %c need to be able to backspace the cursor (cub1), and to move the cursor up one line on the screen (cuu1). This is necessary because it is not always safe to transmit \n, ^D, and \r,asthesystemmay (The library routines dealing with terminfo set tty modes so that tabs are never expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column, offset by a blank character; thus cup=\E=%p1%' '%+%c%p2%' '%+%c . After sending \E=, this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values) and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

If the terminal has row or column absolute cursor addressing, these can be given as single parameter capabilities hpa (horizontal position absolute) and vpa (vertical position absolute). Sometimes these are shorter than the more general two parameter sequence (as with the HP2645) and can be used in preference to cup. If there are parameterized local motions (e.g., move *n* spaces to the right) these can be given as cud, cub, cuf, and cuu with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have cup, such as the TEKTRONIX 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen), this can be given as home; similarly a fast way of getting to the lower left-hand corner can be given as ll; this may involve going up with cuu1 from the home position, but a program should never do this itself (unless ll does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the \EH sequence on HP terminals cannot be used for home.)

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as el. If the terminal can clear from the current position to the end of the display, then this should be given as ed. ed is only defined from the first column of a line (thus, it can be simulated by a request to delete a large number of lines, if a true ed is not available).

Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as `il1`; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, this should be given as `dl1`; this is done only from the first position on the line to be deleted. Versions of `il1` and `dl1` which take a single parameter and insert or delete that many lines can be given as `il` and `dl`. If the terminal has a settable scrolling region (like the vt100) the command to set this can be described with the `csr` capability, which takes two parameters: the top and bottom lines of the scrolling region. Unfortunately, the cursor position is undefined after using this command. It is possible to get the effect of insert- or delete-line using this command. The `sc` and `rc` (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using `ri` or `ind` on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string `wind`. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, the `da` capability should be given; if display memory can be retained below, `db` should be given. These indicate that deleting a line or scrolling can bring non-blank lines up from below or that scrolling back with `ri` can bring down non-blank lines.

Insert or Delete Character

There are two basic kinds of intelligent terminals with respect to insert or delete character that can be described using `terminfo`. The most common insert/delete character operations affect only the characters on the current line, and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen. The character is either eliminated, or expanded to two untyped blanks. To determine what type of terminal you have, clear the screen then type text separated by cursor motions. First, type:

```
abc          def
```

using local cursor motions (not spaces) between the `abc` and the `def`. Next, position the cursor before the `abc` and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters fall off the end, the terminal does not distinguish between blanks and untyped positions. If the `abc` shifts over to the `def` and both strings then move together around the end of the current line and onto the next as you insert, you have the second type of terminal which means that you should give the capability `in`, which stands for *insert null*q. While these are two logically separate attributes (one line vs. multiline insert mode, and special treatment of untyped spaces) it is unlikely you will encounter a terminal whose insert mode cannot be described with the single attribute.

`terminfo` can describe both terminals that have an insert mode, and terminals that send a simple sequence to open a blank position on the current line. Give as `smir` the sequence to get into insert mode. Give as `rmir` the sequence to leave insert mode. Now give as `ich1` any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode do not give `ich1`; terminals that send a sequence to open a screen position should give it here (if your terminal has both, insert mode is usually preferable to `ich1`. Do not give both unless the terminal actually requires that both be used in combination). If post-insert padding is needed, give this as a number of milliseconds in `ip` (a string option). Any other sequence that may need to be sent after an insert of a single character can also be given in `ip`. If the terminal requires being placed in an *insert mode* and also requires that a special code precede each inserted character, then both `smir`/`rmir` and `ich1` can be given, and both will be used. The `ich` capability, with one parameter, *n*, repeats the effects of `ich1` *n* times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line such as when there is a tab after the insertion position). If the terminal allows motion while in insert mode, give the capability `mir` to speed up inserting in this case. Omitting `mir` affects only speed.
Some terminals (notably Datamedia's) must not have `mir` because of the way their insert mode works.

Finally, you can specify `dch1` to delete a single character, `dch` with one parameter, *n*, to delete *n* characters, and delete mode by giving `smdc` and `rmdc` to enter and exit delete mode (any mode the terminal needs to be placed in for `dch1` to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as `ech` with one parameter.

Highlighting, Underlining, and Visible Bells

If the terminal has one or more kinds of display attributes, these can be represented in a number of different ways. Choose one display form as *standout mode*, representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, use **xmc** to indicate how many spaces are left.

Codes to begin underlining and end underlining can be given as **smul** and **rmul** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **blink** (blinking) **bold** (bold or extra bright) **dim** (dim or half-bright) **invis** (blanking or invisible text) **prot** (protected) **rev** (reverse video) **sgr0** (turn off *all* attribute modes) **smacs** (enter alternate-character-set mode) and **rmacs** (exit alternate-character-set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking 9 parameters. Each parameter is either 0 or 1, according to whether the corresponding attribute is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need be supported by **sgr**, only those for which corresponding separate attribute commands exist.

Terminals with the "magic cookie" glitch (**xmc**) deposit special "cookies" when they receive mode-setting sequences that affect the display algorithm, rather than having extra bits for each character. Some terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability is present, asserting that it is safe to move in standout mode.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), this can be given as **flash**; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier-to-find block or blinking underline), give this sequence as **cvvis**. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of both of these modes.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals such as the Concept with more than one page of memory. If the terminal has only memory-relative cursor addressing and not screen relative cursor addressing, a one-screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the TEKTRONIX 4025, where **smcup** sets the command character to be the one used by terminfo.

If the terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, give the capability **ul**. If overstrikes are erasable with a blank, this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1**, **kcuf1**, **kcuu1**, **kcud1**, and **khome** respectively. If there are function keys such as f0, f1, ..., f63, the codes they send can be given as **kf0**, **kf1**, ..., **kf63**. If these keys have labels other than the default f0 through f10, the labels can be given as **lf0**, **lf1**, ..., **lf10**. The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all tabs), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward/down), **kri** (scroll backward/up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1**, **ka3**, **kb2**, **kc1**, and **kc3**. These

keys are useful when the effects of a 3 by 3 directional pad are needed.

Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control I). A "backtab" command which moves leftward to the next tab stop can be given as **cbt**. By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs which are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by the *tset* command to determine whether to set the mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the terminfo description can assume that they are properly set.

Other capabilities include **is1**, **is2**, and **is3**, initialization strings for the terminal, **iprog**, the path name of a program to be run to initialize the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the terminfo description. They are normally sent to the terminal by the *tset* program each time the user logs in. They are printed in the following order: **is1**; **is2**; setting tabs using **tbc** and **hts**; **if**; running the program **iprog**; and finally **is3**. Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is2** and **if**. These strings are output by the *reset* program, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs2** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the vt100 into 80-column mode would normally be part of **is2**, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80 column mode.

If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in **is2** or **if**.

Delays

Certain capabilities control padding in the teletype driver. These are primarily needed by hard copy terminals, and are used by the *tset* program to set teletype modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** cause the appropriate delay bits to be set in the teletype driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used.

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line into which one can cursor address normally (such as the Heathkit h19's 25th line, or the 24th line of a vt100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings to go to the beginning of the status line and to return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The parameter **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to. If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, e.g., **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually control L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, **tparm(repeat_char, 'x', 10)** is the same as **xxxxxxxxxx**.

If the terminal has a settable command character, such as the TEKTRONIX 4025, this can be indicated with **cmdch**. A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. The following convention is supported on some HP-UX systems: The environment is to be searched for a **CC** variable, and if found, all occurrences of the prototype character are replaced with the character in the environment variable.

Terminal descriptions that do not represent a specific kind of known terminal, such as *switch*, *dialup*, *patch*, and *network*, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to *virtual* terminal descriptions for which the escape sequences are known.)

If the terminal uses XON/XOFF handshaking for flow control, give **xon**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted.

If the terminal has a "meta key" which acts as a shift key to set the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software assumes the 8th bit is the parity bit and usually clears it. If strings exist to turn this "meta mode" on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm#0** indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the HP-UX virtual-terminal protocol, the terminal number can be given as **vt**.

Media copy strings, which control an auxiliary printer connected to the terminal, can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal is also sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation **mc5p** takes one parameter, leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local; and **pfx** causes the string to be transmitted to the computer.

Other Considerations

Hazeltine terminals, which do not allow ~ characters to be displayed should indicate **hz**.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the Concept and vt100, should indicate **xenl**.

If **el** is required to get rid of standout (instead of merely writing normal text on top of it), **xhp** should be given.

Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This glitch is also taken to mean that it is not possible to position the cursor on top of a "magic cookie"; thus to erase standout mode it is instead necessary to use delete- and insert-line.

The Beehive Superbee, which is unable to correctly transmit the escape or control C characters, has **xsb**, indicating that the f1 key is used for escape and f2 for Ctrl-C. (Only certain Superbees have this problem, depending on which ROM is installed.)

Other specific terminal problems can be corrected by adding more capabilities of the form **xx**.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be cancelled by placing **xx@** to the left of the capability definition, where **xx** is the capability. For example, the entry

```
2621-nl, smkx@, rmkx@, use=2621,
```

defines a 2621-nl that does not have the **smkx** or **rmkx** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

WARNINGS

HP supports only terminals listed on the current list of supported devices. However, non-supported and supported terminals can be in the terminfo database. If you use such non-supported terminals, they may not work correctly.

FILES

`/usr/lib/terminfo/?/*` files containing terminal descriptions

SEE ALSO

tic(1M), untic(1M), curses(3X), printf(3S), term(4).

Using Curses and Terminfo tutorial in *Terminal Control User's Guide* .

STANDARDS CONFORMANCE

`terminfo`: SVID2

NAME

ttytype - data base of terminal types by port

SYNOPSIS

`/etc/ttytype`

DESCRIPTION

`ttytype` is a database that identifies the kind of terminal that is attached to each tty port on the system. The file contains one line per port, and each line contains the terminal type (as a name listed in *terminfo*(4)), a space, and the name of the tty device file, less the initial `/dev/`. For example, for an HP 2622 terminal on tty02:

```
2622 tty02
```

This information is read by `tset` and by `login` (for remote logins) to initialize the `TERM` variable at login time (see *tset*(1) and *login*(1)).

AUTHOR

`ttytype` was developed by the University of California, Berkeley.

SEE ALSO

login(1), tset(1).

WARNINGS

Some lines are identified simply as `dialup` or `plugboard`.

NAME

tztab - time zone adjustment table for date(1) and ctime(3C)

DESCRIPTION

The **tztab** file describes the differences between Coordinated Universal Time (UTC) and local time. Several local areas can be represented simultaneously with historical detail.

The file **tztab** consists of one or more time zone adjustment entries. The first line of the entry contains a unique string that may match the value of the TZ string in the user's environment. The format is **tzname***diff***dstzname** where **tzname** is the time zone name or abbreviation, *diff* is the difference in hours from UTC, and **dstzname** is the name or abbreviation of the "Daylight Savings" time zone. Fractional values of *diff* are expressed in minutes preceded by a colon. Each such string will start with an alphabetic character.

The second and subsequent lines of each entry details the time zone adjustments for that time zone. The lines contain seven fields each. The first six fields specify the first minute in which the time zone adjustment, specified in the seventh field, applies. The fields are separated by spaces or tabs. The first six are integer patterns that specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), year (1970-2038), and day of the week (0-6, with 0=Sunday). The minute, hour, and month of the year must contain a number in the (respective) range indicated above. The day of the month, year, and day of the week can contain a number as above or two numbers separated by a minus (indicating an inclusive range). Either the day of the month or the day of the week field must be a range, the other must be simple number.

The seventh field is a string that describes the time zone adjustment in its simplest form: **tzname***diff* where **tzname** is an alphabetic string giving the time zone name or abbreviation, and *diff* is the difference in hours from UTC. **tzname** must match either the **tzname** field or the **dstzname** field in the first line of the time zone adjustment entry. Any fractional *diff* is shown in minutes.

Comments begin with a # in the *first* column, and include all characters up to a new-line. Comments are ignored.

If the value of the **TZ** string does not match any line in the table, it is interpreted according to the current U.S. pattern.

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

EXAMPLES

The time zone adjustment table for the Eastern Time Zone in the United States is:

```
EST5EDT
0 3 6      1  1974       0-6 EDT4
0 3 22-28  2  1975       0   EDT4
0 3 24-30  4  1976-1986  0   EDT4
0 3 1-7    4  1987-2038  0   EDT4
0 1 24-30  11 1974       0   EST5
0 1 25-31  10 1975-2038  0   EST5
```

Normally (as indicated in the first line) Eastern Standard Time is five hours earlier than UTC. During Daylight Savings time, it changes to a 4 hour difference. The first time Daylight Savings Time took effect (second line) was on January 6, 1974 at 3:00 a.m., EDT. Note that the minute before was 1:59 a.m., EST. The change back to standard time took effect (sixth line) on the last Sunday in November of the same year. At that point, the time went from 1:59 a.m. EDT to 1:00 a.m. EST. The transition to Daylight Savings Time since then has gone from the last Sunday in February (third line) to the last Sunday in April (fourth line) to the first Sunday in April (fifth line). The return to standard time for the same period has remained at the last Sunday in October (seventh line).

AUTHOR

tztab was developed by HP.

FILES

/usr/lib/tztab

SEE ALSO
date(1), ctime(3C), environ(5).

NAME

update - update-media format

DESCRIPTION

Tape update media consist of a simple **tar** archive (see *tar*(1)) with a few leading information files used by **update** (see *update*(1M)) plus specially-crafted file paths that allow files to be grouped into filesets. The following is intended as an aid in interpreting tape update media; not as a design guide for building your own. Update script format is also documented here.

On a netdist server system and on a CD-ROM, the update "media" is a collection of files in a directory hierarchy (exact format varies according to media type), plus various information files.

Information Files

The information files contain ASCII text and include:

system/INDEX

This file describes the filesets on the media. It consists of a header line followed by a series of blocks (paragraphs) of data (text), one for each fileset on the media. Each block is bounded by "**begin:**" and "**end:**" and contains labeled attributes for the fileset. The file contains lines of the form:

INDEX:	media_number media_version_number checksum_algorithm_number
begin:	fileset_name
mn:	medium_number
fd:	fileset_description
pn:	partition_name
pd:	partition_description
ff:	fileset_flags (see Fileset Flags below)
is:	instruction_sets (A.B8.05 media only; see Fileset Types below)
sys:	system_types (A.B8.05 media only; see Fileset Types below)
fs:	fileset_size (in bytes; normally computed by *update*)
fv:	fileset_version_number
dep:	dependee_fileset_1 dependee_version_number (minimum needed)
dep:	dependee_fileset_2 dependee_version_number
...	
ffile:	INFO_file_information
...	
frule:	CDFinfo_file_information
...	
end:	fileset_name
begin:	fileset_name
...	(similar information about next fileset)
end:	fileset_name
ENDINDEX:	checksum

Dependency lines include both direct and indirect dependencies. The **update** and **updist** commands do not recursively apply dependencies.

On multiple-tape update media, the INDEX file on each tape is identical to that on all other tapes except for the media unit number in the header line.

system/INFO

This file describes the files on the media. For each fileset on the media, it contains a list of all the files in the fileset along with their sizes and special attributes. The file contains lines of the form:

INFO:	media_version_number	checksum_algorithm_number
#Fileset	**Bytes**	**File**
#==========	**========**	**==========**
fileset_name		
+type[,attr]	size (bytes)	file_name [link_target]
+type[,attr]	size	file_name [link_target]

```
...

fileset_name
+type[,attr]        size        file_name [link_target]
+type[,attr]        size        file_name [link_target]
...
ENDINFO: checksum
```

Some HP-UX file attributes cannot be contained in a CD-ROM file system. They are included in the **INFO** file instead, in the *type* and *attr* fields. The value of the *type* field is one of the following. These are the only file types supported on any update media.

RF	regular file
SL	symbolic link (requires link_target)
HL	hard link (requires link_target)
DR	directory

The *attr* field is an octal value representing the file's mode bits. The value from the source media is used, and the **INFO** file value is ignored, except when loading from CD-ROM media. File mode bits (from the source media or the *attr* field) are ignored for symbolic and hard links (**SL** and **HL** entries), and left unchanged for directories (**DR** entries) except when a new directory is created.

The *size* field is the string length of *link_target* for **SL** entries, and zero for **HL** and **DR** entries.

On multiple-tape update media, the **INFO** file on each tape is identical to that on all other tapes.

system/CDFinfo

This file contains a list of stand-alone file path names and one or more rules concerning each path name. The rules describe changes associated with each file when turning a standalone system into a cluster server, updating a cluster server, or adding a cnode to a clustered system. The **sam** and **update** commands (see *sam*(1M) and *update*(1M)) apply these rules during such operations. Only files whose paths require changes during cluster operations appear in the **CDFinfo** file.

If update media lacks **CDFinfo** files, **update** issues a warning, but **updist** refuses to load from the media because the result might be a netdist server on which some filesets have CDF information and others do not. (The workaround is to put empty **CDFinfo** files on the media for each fileset.)

Fileset Flags

Any of five flags can be associated with each fileset. Their meanings are:

B Rebuild the kernel and reboot the system.

C Change of destination is not allowed for this fileset. The fileset must be loaded under **/**.

M or **H**
Fileset contains HP-MC68020 (Series 300/400) files or HP-PA (Series 700 or Series 800) files, respectively. Beginning with release 8.05, the **M** flag is equivalent to:

```
is:  MC68020
sys:  S300
```

(see Fileset Types below). The **H** flag is equivalent to:

```
is:  PA_RISC_1_0
sys:  S700, S800
```

Note that the **M** and **H** flags and the **is** and **sys** fields are mutually exclusive. Each fileset's type must be specified with a flag or the **is** and **sys** fields, but not both. Beginning with A.B8.05-format media, the latter form is preferred.

S
Fileset is secured (protected) against direct access from a CD-ROM.

Y
Use **rmfn** to remove the files in the fileset before loading new ones (see *rmfn*(1M)). Use of this flag is discouraged because processing it is time consuming. The removal is better done for specific cases by

customize scripts.

The B, C, and Y flags are used by `update` and ignored by `updist`.

Fileset Types on A.B8.05 Format Media

Beginning with the 8.05 release (media format A.B8.05), the `is` field specifies the instruction set(s) types of the system CPU on which the fileset can be loaded. The field value is one of:

`MC68020`	Motorola 68020, 68030, or 68040
`PA_RISC_1_0`	HP-PA RISC 1.0 or 1.1
`PA_RISC_1_1`	HP-PA RISC 1.1
*	Indicates no particular instruction set is required. For example, this typically marks a fileset conatining only shell scripts and simple text.

The `sys` field specifies the system type(s) on which the fileset can be loaded. The field value is one or more comma-separated values from this list.

`S300`	Series 300 or 400
`S700`	Series 700
`S800`	Series 800
*	Indicates the fileset is intended for all supported systems.

For example, consider a program compiled to use HP-PA RISC 1.0 instructions that does not distinguish Series 700 and Series 800 system types, or which handles the distinctions at run time. A fileset containing this program might be marked:

```
is:  PA_RISC_1_0
sys:  S700, S800
```

Path Names

On tape update media, path names of files other than the information files consist of their official path names with a concatenated prefix:

fileset`/..`

For example:

```
UX-CORE/../bin/sh
```

These path names indicate each file's fileset and still allow unpacking with `tar` relative to any directory. Unpacking a tape update media unit with `tar` rather than `update` is discouraged, partly because it creates empty directories on the system, one for each fileset on the update media. The empty directories can be removed.

Each fileset's files appear on the media grouped together (in a contiguous sequence).

Update Scripts

Update script files written and read by `update` (see *update*(1M)) have the following format:

- Blank and comment lines are ignored. Comment lines have # as the first non-whitespace character.
- All other lines are data lines and must end in \. Any whitespace at the start of the line or before the backslash is ignored.
- Each data line consists of one of the following forms.

  name='value' \
  command_name \
  -option_letter \
  -option_letter 'option_argument' \
  fileset_name \

- Any variable declarations (*name='value'*) must appear before any other line types. The only supported environment variable in this context is `UPDATENOSKIP`.

- The first data line after variable declarations (if any) must be the full path name of the command as invoked from the command line, and it must match the current invocation name.
- Next are command options, one per line, each starting with -, with option arguments where required. Update scripts support any of the **-msSPCdrb** options only; options **-Ffic** cannot appear in them. For a netdist server source, the **-P** option must be present. For a CD-ROM source, the **-C** option must be present. (In this context the presence of these options also specifies the source type as other than tape.)
- Options are followed by zero or more fileset names, one per line, to cause pre-selection of filesets. Either fileset names or the **-m** option can be present, but not both, and neither is required. Note that **-m** causes matching on the system where the update script is run or read, possibly with different results than on the system where the script was written.

Here is an example of an update script.

```
UPDATENOSKIP='0' \
/etc/update \
-s 'hpfclc.fc.hp.com' \
-S '300' \
-P '8080' \
-d '/' \
-r \
KERN-BLD \
LAN \
LSSERVER-ADMIN \
TOOL \
```

EXAMPLES

List the files on a tape update media unit accessed through special file **/dev/rmt**:

```
tar -tvf /dev/rmt
```

Display the leading information files on a given tape update media unit:

```
cd /tmp
tar -xvf /dev/rmt system
more system/*
```

DEPENDENCIES

Series 700

For HP-UX release 8.01, unlike release 8.05, **update** and **updist** do not distinguish Series 700 and Series 800 update media. Media for both Series are version A.B8.00 and bear the fileset flag **H**.

SEE ALSO

tar(1), rmfn(1M), sam(1M), update(1M).

NAME

utmp, wtmp, btmp - utmp, wtmp, btmp entry format

SYNOPSIS

```
#include <sys/types.h>
#include <utmp.h>
```

DESCRIPTION

These files, which hold user and accounting information for such commands as `last`, `who`, `write`, and `login` (see *last*(1), *who*(1), *write*(1), and *login*(1)), have the following structure as defined by <utmp.h>:

```
#define UTMP_FILE    "/etc/utmp"
#define WTMP_FILE    "/etc/wtmp"
#define BTMP_FILE    "/etc/btmp"
#define ut_name ut_user

struct     utmp {
    char     ut_user[8];              /* User login name */
    char     ut_id[4];                /* /etc/inittab id (usually line #) *
    char     ut_line[12];             /* device name (console, lnxx) */
    pid_t    ut_pid;                  /* process id */
    short    ut_type;                 /* type of entry */
    struct  exit_status {
        short    e_termination;       /* Process termination status */
        short    e_exit;              /* Process exit status */
        } ut_exit;                    /* The exit status of a process */
                                      /* marked as DEAD_PROCESS. */
    unsigned short  ut_reserved1;     /* Reserved for future use */
    time_t  ut_time;                  /* time entry was made */
    char     ut_host[16];             /* host name, if remote */
    unsigned long   ut_addr;          /* Internet addr of host, if remote *,
};

/*  Definitions for ut_type  */
#define    EMPTY 0
#define    RUN_LVL    1
#define    BOOT_TIME  2
#define    OLD_TIME   3
#define    NEW_TIME   4
#define    INIT_PROCESS     5     /* Process spawned by "init" */
#define    LOGIN_PROCESS    6     /* A "getty" process waiting for login */
#define    USER_PROCESS     7     /* A user process */
#define    DEAD_PROCESS     8
#define    ACCOUNTING 9
#define    UTMAXTYPE  ACCOUNTING /* Largest legal value of ut_type */

/*  Special strings or formats used in the "ut_line" field when  */
/*  accounting for something other than a process  */
/*  No string for the ut_line field can be more than 11 chars +  */
/*  a NULL in length  */
#define RUNLVL_MSG    "run-level %c"
#define BOOT_MSG      "system boot"
#define OTIME_MSG     "old time"
#define NTIME_MSG     "new time"
```

File `btmp` contains bad login entries for each invalid logon attempt.

Note that `wtmp` and `btmp` tend to grow without bound, and should be checked regularly. Information that is no longer useful should be removed periodically to prevent it from becoming too large.

In the HP Clustered environment, the files **utmp**, **wtmp**, and **btmp** are context-dependent files (CDFs). See *cdf*(4). These files must be CDFs so that the boot-time and run-level entries represent the actual state of each cluster cnode.

FILES

```
/etc/utmp
/etc/wtmp
/etc/btmp
```

AUTHOR

utmp, **wtmp**, and **btmp** were developed by HP and the University of California, Berkeley.

SEE ALSO

last(1), login(1), who(1), write(1), acctcon(1M), fwtmp(1M), getut(3C), cdf(4).

STANDARDS CONFORMANCE

<utmp.h>: **XPG2**

NAME

uuencode - format of an encoded uuencode file

DESCRIPTION

Files output by **uuencode** consist of a header line followed by a number of body lines, and a trailer line. The **uudecode** command ignores any lines preceding the header or following the trailer (see *uudecode*(1)). Lines preceding a header must not look like a header.

The header line consists of the word **begin** followed by a space, a mode (in octal), another space, and a string which specifies the name of the remote file.

The body consists of a number of lines, each containing 62 or fewer characters (including trailing new-line). These lines consist of a character count, followed by encoded characters, followed by a newline.

The character count is a single printing character, which represents an integer. This integer is the number of bytes in the rest of the line, and always ranges from 0 to 63. The byte count can be determined by subtracting the equivalent octal value of an ASCII space character (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a space to make the characters printable. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra meaningless data will be included, if necessary, to make the character count a multiple of 4. The body is terminated by a line with a count of zero. This line consists of one ASCII space.

The trailer line consists of the word **end** on a line by itself.

SEE ALSO

mail(1), uudecode(1), uuencode(1), uucp(1), uusend(1).

NAME

uuidname.txt - file associating names with UUIDs

SYNOPSIS

Public file:

`/sys/ncs/uuidname.txt` (SR9 Apollo Aegis workstations)
`/etc/ncs/uuidname.txt` (SR10 Apollo Domain/OS workstations)
`/etc/ncs/uuidname.txt` (HP-UX systems and other UNIX systems)
`ncs$exe:uuidname.txt` (VMS systems)
`\ncs\uuidname.txt` (MS-DOS systems)

Private file:

`~/uuidname.txt` (Domain/OS, HP-UX, and other UNIX systems)

DESCRIPTION

A `uuidname.txt` file associates textual names with UUIDs (Universal Unique Identifiers). The `lb_admin` administrative tool can use these names to identify objects, types, and interfaces (see *lb_admin*(1M)); it accepts names as input and displays names as output whenever possible.

System-wide associations of names with UUIDs are defined in a public `uuidname.txt` file on the host where `lb_admin` is invoked. On Apollo Domain/OS workstations, HP-UX systems, and other UNIX systems, user-specific associations can also be defined in a private `uuidname.txt` file in the home directory of the user who invokes `lb_admin`. `lb_admin` reads these files (first the public file then the private file, if it exists) when it starts up, and uses the names defined in the files for the duration of the session.

Each UUID in a `uuidname.txt` file appears at the beginning of a line. Names associated with that UUID occupy the remainder of the line, separated by spaces or tabs. Names that contain spaces or tabs must be delimited by double quotation marks. Blank lines and lines beginning with `#` are ignored. A `#include` construct supports inclusion of other files in this format.

More than one name can be associated with a UUID if several names appear on one line of a `uuidname.txt` file, if a UUID appears on several lines of one file, or if a UUID appears in several files. The first name encountered by `lb_admin` when it starts up is treated as the "primary name" for the UUID, and all subsequent names are treated as "aliases". Any primary names or aliases can be entered as input to `lb_admin`, but the tool always uses primary names for output.

If an undefined name is entered as input, `lb_admin` treats the input as a wildcard.

Note that this mechanism for associating names with UUIDs may be superseded by a more general naming service in a future software release.

EXAMPLE

The following is part of a sample `uuidname.txt` file:

```
333b91c50000.0d.00.00.87.84.00.00.00    glb/object
333b91de0000.0d.00.00.87.84.00.00.00    glb/type
333b2e690000.0d.00.00.87.84.00.00.00    glb/interface
34b45208a000.0d.00.00.87.84.00.00.00    rgy/object
```

SEE ALSO

lb_admin(1M).

Managing NCS Software.

NAME

vhe_list - information file for the Virtual Home Environment

DESCRIPTION

/etc/vhe_list is an ASCII file that contains the information needed to configure a group of machines together with the Virtual Home Environment (VHE). These machines are connected using the Network File System (NFS). The information from **vhe_list** is used by the script **vhe_mounter**.

An entry in **vhe_list** contains the following information:

- Host name of a machine exporting a file system.
- Name of the file system to be mounted by NFS.
- Name of the directory that acts as the mount point.
- Mount options for the NFS mount (this is optional).

For every file system that is to be available (exported) for NFS mounting for VHE, there is an entry in the **vhe_list** file. Blank lines, lines of white spaces, or lines beginning with the # character are ignored.

EXAMPLES

Consider two machines named **high** and **low**, each to be connected with VHE. Machine **high** is exporting the file system **/** to be mounted on directory **/vhe/high**. Machine **low** is exporting the file system **/** to be mounted on directory **/vhe/low** and the file system **/users** to be mounted on directory **/vhe/low/users** using the NFS mount options of **timeo=10,wsize=4096**. For this situation, the contents of the **vhe_list** file would resemble the following:

```
high      /          /vhe/high
low       /          /vhe/low           timeo=10,wsize=4096
low       /users     /vhe/low/users     timeo=10,wsize=4096
# A comment line
```

Mount options must be separated by commas, and must contain no spaces. Mount options are the same as those used in the **mount** command (see *mount*(1M)).

AUTHOR

vhe_list was developed by HP.

FILES

/etc/vhe_list

SEE ALSO

vhe_altlog(1M), vhe_mounter(1M), vhe_u_mnt(1M).

(Requires Optional LAN/X.25 Software)

NAME

x25_networks - identifies the network types used by the system

DESCRIPTION

The network type file contains information describing the several X.25 public network types. A **x25_networks** file is supplied as part of the X.25/9000 product in directory **/etc/x25**. Entries in that file can be modified to suit individual system needs.

x25_networks contains up to 5 columns in the following order:

1. An alias which can be specified for the *networktype* parameter in the **x25init** configuration file.

   Aliases appearing in this column are user-defined strings, and can consist of any character except the hash character (#), the space character, or newline. The alias must contain not more than 30 characters.

2. Network type. The network type must be one of the network names that appear in column two of the supplied file.

   If adding an entry to this file, use the "generic" networks which are defined in the original **x25_networks** file (**L3_DTE** or **L3_DCE**).

   If creating an entry for a variation of one of the standard public networks already recognized (such as TransPac), use that network's network type. If a network type is not specified, the default value of **DTE_84** which indicates a standard DTE that complies with the 1984 X.25 recommendation is assumed.

3. Version of the CCITT X.25 recommendation that the network honors. Only two values are allowed: either **1980** or **1984**, designating which version of the CCITT X.25 recommendations the network uses.

4. Refers to network behavior and is optional. This column can contain one of 3 entries: **NOFACCHK**, **NODUPFAC**, or **DDN**.

   NOFACCHK indicates that no facilities checking will be done on the facilities field. If **NOFACCHK** is not specified, the facilities are checked to ensure that they comply with the appropriate CCITT recommendations (**1980** or **1984**).

   NODUPFAC indicates that a facility code cannot be specifed twice in the same facility/registration code field. This affects facilities such as **closed user group** in which only one facility may be specified; that is, incoming calls can be barred or outgoing calls can be barred, but not both.

   DDN indicates that the Maximum Transmission Unit (MTU) size is to be set to 1007, and the **DDN** address mapping algorithm is to be used instead of the IP-to-X.121 address map table. The DDN address mapping is defined in the *DDN X.25 Host Interface Specification (BBN83)*.

5. Also refers to network behavior and is also optional. This column can contain a second entry from the fourth column: **NOFACCHK**, **NODUPFAC**, or **DDN**.

EXAMPLES

Here are some example lines from the **x25_networks** file:

```
DCE_84           L3_DCE          1984    NODUPFAC
DATANET1         DATANET1        1984    NODUPFAC
DATAPAC          DATAPAC         1980    NODUPFAC
DATEXP_Austria   DATEXP_AUSTRIA  1984    NODUPFAC
```

WARNINGS

NOFACCHK and **NODUPFAC** are incompatible and cannot be specifed on the same line.

AUTHOR

x25_networks was developed by HP.

SEE ALSO

X25init(1M),

Installing and Administering X.25/9000.

(Requires Optional LAN/X.25 Software)

NAME

x25init_smpl - sample configuration file used to initialize an X.25 interface.

DESCRIPTION

To simplify and automate X.25 interface configuration, an ASCII file containing some or all X.25 interface configuration parameters and values can be created for use by the **x25init** command (see *x25init*(1M)). **x25init** then uses the file's contents to load the configuration data structures in the X.25 interface. A sample configuration file, **x25init_smpl**, is distributed with the system, and can be customized to suit the needs of each installation. To configure the interface, use the command form:

x25init -c*config_file*

See *x25init*(1M) for more information about command usage.

The X.25 configuration file is arranged in a command-like structure such that configuration parameters and their assigned values are listed, line-by-line, in an easily understood format. Individual entries in the file can appear in any order, but only one entry is allowed on each line. Many parameters can be specified using an abbreviated form.

Syntax

Entires in the configuration file have the form:

parameter_name parameter_value

Where:

parameter_name Specifies a keyword identifying a particular configuration parameter. Some examples are **x121**, **device**, and **framesize**.

parameter_value Specifies the desired value for the corresponding parameter.

Blank lines, leading spaces, and trailing spaces are ignored. The # character and anything that follows it on a given line is treated as a comment. If a parameter is specified more than once in the configuration file, the value of the last entry for that parameter is used.

Parameters

X.25 configuration parameters fall into the following categories:

Required: Required parameters must be specified, and include:

X.121 address
device file
X.25 Programmatic Access name

Level 2:
Used to define X.25 Level 2 (data link layer) characteristics:

framesize
level-2 window
retransmission count
T1 timer
T3 timer

Level 3:
Used to define X.25 level 3 (network layer) characteristics:

circuit table definition
default outbound packet size
default outbound throughput size
default outbound window size
network type
negotiated inbound packet size
negotiated inbound throughput class
negotiated inbound window size
reverse charge accept
PVC inbound packet size
default inbound packet size
default inbound throughput size
default inbound window size
flow control negotiation
fast select accept
negotiated outbound packet size
negotiated outbound throughput class
negotiated outbuond window size
X.121 packet[addr]
PVC outbound packet size

(Requires Optional LAN/X.25 Software)

PVC inbound throughput class
PVC inbound window size
throughput class negotiation

PVC outbound throughput class
PVC outbound window size

IP Related:
Used to define the IP-related characteristics:

hold timer
internet (IP) address
maximum circuits

idle timer
maximum transmission unit

Parameter Definitions

The following configuration parameters and values are recognized:

lci *start_num type how_many*
(Level 3 parameter) Specifies the circuit table definition for the system; includes the starting value of the logical channel identifier, the type of virtual circuits, and the number of each type.

def_inpacket [size] *size*
(Level 3 parameter) Specifies the default maximum size, in octets, of a level-3 packet when flow control negotiation is off.

def_inthruput [class] *class*
(Level 3 parameter) Specifies the maximum allowable inbound data transmission rate in bits per second on a switched virtual circuit when throughput class is not negotiated.

def_inwindow *size*
(Level 3 parameter) Specifies the default maximum number of unacknowledged inbound packets between the DTE and the DCE when flow control negotiation is off.

def_outpacket [size] *size*
(Level 3 parameter) Specifies the default maximum size, in octets, of a level-3 outbound packet when flow control negotiation is off.

def_outthruput [class] *class*
(Level 3 parameter) Specifies the maximum allowable outbound data transmission rate in bits per second on a switched virtual circuit when throughput class is not negotiated. The value of *class* can be expressed in bits per second: 75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 48000, or the corresponding CCITT class numbers: 3, 4, 5, 6, 7, 8, 9, 10, 11, or 12.

def_outwindow *size*
(Level 3 parameter) Specifies the default maximum number of unacknowledged outbound packets between the DTE and the DCE when flow control negotiation is off.

device *name*
(Required parameter) Specifies the interface card driver device file name found in the **/dev** directory.

fast_select_accept enabled
fast_select_accept disabled
(Level 3 parameter) Specifies whether or not the fast select facility is allowed. When fast select is enabled, up to 128 octets of user data can be sent with the call set-up and clear packets.

flow[control] on
flow[control] off
(Level 3 parameter) Specifies flow control negotiation, which involves choosing the level-3 packet size and level-3 window size.

frame [size] *n*
n1
(Level 2 parameter) Specifies the maximum number of octets that can be transmitted in one frame. Either **framesize** or **n1** can be specified as the parameter name.

(Requires Optional LAN/X.25 Software)

hold [timer] *timer* (IP-related parameter) Specifies the hold timer in seconds.

idle [timer] *timer* (IP-related parameter) Specifies the idle timer in seconds.

IP [address] *address* [*subnet_mask*]
(IP-related parameter) Specifies the Internet address for the system if IP is to be used. The Internet address is expressed in dot notation.

l2window
k *size* (Level 2 parameter) Specifies the maximum number of unacknowledged frames between the DTE and the DCE. Either **l2window** or **k** can be specified as the parameter name.

max_circuit *count* (Level 3 parameter) Specifies the maximum number of virtual circuits that can be open at one time.

mtu *size* (IP-related parameter) Specifies the maximum number of octets that can be transmitted in one IP packet.

neg_inpacket [size] *size*
(Level 3 parameter) Specifies the maximum size, in bytes, of a level-3 inbound packet when flow control negotiation is on.

neg_inthruput [class] *class*
(Level 3 parameter) Specifies the maximum allowable inbound data transmission rate in bits per second on a switched virtual circuit when throughput class is negotiated. The value of class can be expressed in bits per second: 75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 48000, or the corresponding CCITT class numbers: 3, 4, 5, 6, 7, 8, 9, 10, 11, or 12.

neg_inwindow *size* (Level 3 parameter) Specifies the maximum number of unacknowledged inbound packets between the DTE and DCE when flow control negotiation is turned on.

neg_outpacket [size] *size*
(Level 3 parameter) Specifies the maximum size, in octets, of a level-3 outbound packet when flow control negotiation has been turned on.

neg_outthruput [class] *class*
(Level 3 parameter) Specifies the maximum allowable outbound data transmission rate in bits per second on a switched virtual circuit when throughput class is negotiated. The value of class can be expressed in bits per second: 75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 48000, or the corresponding CCITT class numbers 3, 4, 5, 6, 7, 8, 9, 10, 11, or 12.

neg_outwindow *size* (Level 3 parameter) Specifies the maximum number of unacknowledged outbound packets between the DTE and the DCE when flow control negotiation has been turned on.

networktype *user_defined_network_name*
(Level 3 parameter) Specifies the network interface type. The network type specified must match the line configuration of the host. Network type names and their associated characteristics are specified in an editable file used to identify the network to the subsystem.

name *name* (Level 3 parameter) Supplies the programmatic access name used for programmatic access to X.25 at level 3.

pvc_inpacket [size] *size*
(Level 3 parameter) Specifies the maximum size, in octets, of a level-3 inbound packet on a permanent virtual circuit.

pvc_inthruput [class] *class*
(Level 3 parameter) Specifies the maximum allowable inbound data transmission rate in bits per second on a permanent virtual circuit. The value of class is expressed in either bits per second: 75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, or the corrsponding CCITT class numbers: 3, 4, 5, 6, 7, 8, 9, 10, 11, or 12.

pvc_inwindow *size* (Level 3 parameter) Specifies the maximum number of unacknowledged inbound packets between a DTE and a DCE on a permanent virtual circuit.

pvc_outpacket [size] *size*
(Level 3 parameter) Specifies the maximum size, in octets, of a level-3 outbound packet on a permanent virtual circuit.

pvc_outthruput [class] *class*
(Level 3 parameter) Specifies the maximum allowable outbound data transmission rate in bits per second on a permanent virtual circuit. The value of class can be expressed in either bits per second: 75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, or the corresponding CCITT class numbers: 3, 4, 5, 6, 7, 8, 9, 10, 11, or 12.

pvc_outwindow *size* (Level 3 parameter) Specifies the maximum number of unacknowledged outbound packets between DTE and DCE on a permanent virtual circuit.

n2 *count* (Level 2 parameter) Specifies the number of times a frame can be retransmitted when an error occurs.

reverse_charge_accept enabled
reverse_charge_accept disabled
(Level 3 parameter) Specifies whether or not reverse charging and reverse charging accepted are supported. A DTE that subscribes to reverse charging can request, on a per-call basis, that the remote DTE pay for the call. Reverse charging acceptance authorizes the driver to pass to the application any calls requesting reverse charging. If this facility is disabled, the local DCE rejects any incoming calls that request reverse charging.

t1 *time* (Level 2 parameter) Specifies the amount of time to wait for acknowledgement of a frame.

t3 *time* (Level 2 parameter) Specifies the idle time.

thruputclass on
thruputclass off (Level 3 parameter) Specifies the throughput class negotiation between peers.

X.121*address* (Required parameter) Specifies the X.121 address of your system. The value of *address* can be up to 15 digits.

X.121_packet[*addr***]** (Level 3 parameter) Specifies the X.121 packet address of any network of which there are special characteristics. An example of some network types are TRANSPAC, DDN and TYMNET. When using these and other networks known to require this parameter, the value should be expressed as 15 digits enclosed in single quotes. For these networks, a value should be specified for this parameter in addition to the standard X.121 address parameter.

DIAGNOSTICS

If an error is encountered while parsing the commands in a configuration file, **x25init** reports the error and continues parsing the remainder of the file. At the end of the file, **x25init** exits without initializing the X.25 interface.

EXAMPLES

Here are two entries as they might appear in a typical configuration file:

```
X.121 4085551212
framesize 256
```

DEPENDENCIES

Use of the parameters **fast_select_accept** and **reverse_charge_accept** is dependent upon subscription to these facilities from the network provider.

AUTHOR

x25init_smpl was developed by HP.

NOTES

The configuration file can be built interactively through use of the **x25init** interactive user interface.

(Requires Optional LAN/X.25 Software)

Refer to *x25init*(1M) for an explanation of how to use the `x25init` interface.

SEE ALSO

X25init(1M),
Installing and Administering X.25/9000,
Troubleshooting X.25/9000.

(Requires Optional LAN/X.25 Software)

NAME

x29hosts - PAD-related configuration file

DESCRIPTION

x29hosts is an ASCII file in directory **/etc/x25** that provides configuration information for the X.25 PAD components: **padem**, **x29printd**, **x29server**, and **x29uucpd**. Although the **x25init** command is used to configure the interface cards as well as the X.25 subsystem, further configuration is required for the use of the PAD components.

/etc/x25/x29hosts is made up of one or more parameter groups. The syntax of the parameter groups in the file is as follows:

```
component_type {
    device              device_name
    name                programmatic_access_name
    remote_x121         remote_x121_address
    x3                  x3_config_set_name
    profile             profile_ID
    reverse_charge      request
    logging             level
}
```

Items in **computer font** are parameter names and are required if a corresponding parameter value is given. A parameter value follows the parameter name on each line. Parameter name-value pairs can be arranged in any order, but each must appear on a separate line, terminated by a new-line character. Any amount of whitespace (spaces or tabs) can be inserted as long as each parameter group consists of the *component_type* followed by a pair of curly braces enclosing the description parameters, each on a separate line as shown above. Description parameters are constructed as follows:

Component Type — syntax: *component_type*

The component type field is checked during initialization of each PAD application program. Each program extracts only those parameter groups that are relevant to its own operation. Recognized component types are:

printer	(used by **x29printd**)
pad_spt	(used by **x29server**)
pad_uucp	(used by **x29uucpd**)
pad_em	(used by **padem**)

Device Name

syntax: **device** *device_name*

The device name defines which device file in directory **/dev/x29** is to be used. This field must be present if *component_type* is either **printer** for **x29printd** or **pad_uucp** for **x29uucpd**. This field is not meaningful for **pad_em** and **pad_spt**.

X.25 Programmatic Access Name

syntax: **name** *programmatic_access_name*

The programmatic access name defines the interface card to use for outbound calls. It is the same name as used by the **x25init** command (see *x25init*(1M)). This field is not meaningful for **pad_spt**, which accepts calls on all interfaces.

Remote X.121 Address

syntax: **remote_x121** *remote_x121_address*

The X.121 address field defines the remote address to which this PAD component will connect. It can be an extended address based on the address of the interface port that the remote PAD device is using.

X.3 Configuration

syntax: **x3** *x3_config_set_name*

The X.3 configuration set name refers to the name given to the set of X.3 parameters used by this PAD component for vitual circuits to this X.121 address. The X.3 configuration sets are found in the file **/etc/x25/x3config**. This field is only meaningful for **pad_spt**, **printer**, and **pad_uucp**.

Logging Level
syntax: **logging** *level*

The logging field defines the severity of messages to be saved either in the files under directory **/usr/adm/x29** (for **x29printd**, **x29uucpd**, and **x29server**) or in **$HOME/plog.***PID* (for **padem**).

Profile
ID syntax: **profile** *profile_ID*

profile_ID is an integer that can be used in adition to the X.3 configuration set name. *profile_ID* is a reference to a defined set of X.3 parameters found in file **/etc/x25/x3config**. This field is only meaningful for **padem**.

Reverse Charging
syntax: **reverse_charge** *request*

This field tells the PAD component whether reverse charging is requested in the call request packet for the PAD components **x29printd**, **x29uucpd**, and **padem**. It specifies whether reverse charge is allowed for specific incoming calls if it is used in the context of **pad_spt**. Valid *request* values are **enable**, **enabled**, **disable**, and **disabled**. If this parameter is not specified, the default value is **disabled**.

Also found in the **x29hosts** file is the host table, which is a mechanism for mapping X.121 addresses to symbolic hostnames manipulated by the end-user. The syntax for the host table is:

```
host_table {
        [symbolic_hostname address]
                ...
                ...
}
```

host_table must be the first token to appear in the host table entry. Each entry in the table consists of a *symbolic_hostname* followed by a corresponding *adddress*. *symbolic_hostname* must be listed first, followed by one or more tabs or spaces, then the X.121 address. Each table entry must appear on a separate line, terminated by a new-line character. The group of one or more entries in the table must be enclosed within a pair of curly braces {}. Comments are allowed in the file and have the character # at the beginning of the line. **host_table{}** is only used by **padem**.

EXAMPLES

```
# Sample x29hosts file

printer {
    device printer1
    name hptndxk0
    remote_x121 408555111201
    x3 hp_printer
    reverse_charge enable
    logging 1
}

pad_uucp {
    device x25uucp
    name hptndxk0
    remote_x121 4085551113
    x3 hp_uucp
    reverse_charge enable
    logging 3
}

host_table {
    Gale 4085551111
    Tornado 4085551113
    Typhoon 40855511115
}
```

(Requires Optional LAN/X.25 Software)

```
pad_em {
    name hptndxk0
    remote_x121 4085551111
    reverse_charge enable
    profile 0
    logging 3
}
pad_spt {
    remote_x121 408555120801
    x3 hp_padsrvr
    logging 1
    reverse_charge disable
}
```

End of sample x29hosts file

AUTHOR

x29hosts was developed by HP.

SEE ALSO

padem(1), x25init(1M), x29printd(1M), x29server(1M), x29uucpd(1M), x3config(4).

Installing and Administering X.25/9000.

(Requires Optional LAN/X.25 Software)

NAME

x3config - PAD-related X.3 configuration file

DESCRIPTION

x3config is an ASCII file in directory **/etc/x25** that contains X.3 configuration information for X.25 PAD (packet assembler/disassembler) components **padem**, **x29printd**, **x29server**, and *x29uucpd*. **x3config** contains many different terminal-specific characteristics or actions that the hosts want the remote PAD, PAD support, or PAD emulation to take upon receipt of specific input from the hosts or from the remote PAD device. **x3config** contains different configuration groups that the user can choose to use as initial X.3 parameters. The groups of X.3 parameters are referenced by X.3 configuration set names in the file **/etc/x25/x29hosts**.

x3config contains the following information:

- X.3 Configuration Set Name
- X.3 Parameter
- Corresponding Parameter Value
- Comment Lines

The **x3config** file is made up of one or more X.3 configuration sets. Each X.3 configuration set has its own configuration entry consisting of X.3 parameters and corresponding values (each parameter-value pair on a separate line) enclosed in curly braces {}, and preceeded by the X.3 configuration set name (and profile). The X.3 configuration set name is defined by the administrator, and maps to the PAD configuration entries in file **/etc/x25/x29hosts**. The administrator can also define profile identification numbers *profile_id* s for X.3 configurations. *prof_id* is an integer number that is used by *padem* as the method of referencing X.3 configurations. *prof_id* immediately follows *x3_config_set_name* and a colon (:). A typical X.3 **x3config** configuration set is constructed as follows:

```
x3_config_set_name {
# <X.3 parameter> <value1> <value2>
          1            1        1
          2            1        1
          .            .        .
          .            .        .
          .            .        .
         22            0        0
}
```

Parameter interpretation is as follows:

x3_config_set_name A string that identifies the accompanying set of parameter-value pairs enclosed within curly braces. Each X.3 configuration set name should be unique. *x3_config_set_name* must be concatenated with a profile ID when defining the X.3 configuration for **padem** (see *padem*(1)).

X.3 Parameter An integer whose value ranges from 1 through 22 and defines the type of X.3 parameter the user is requesting.

Parameter Value Assigns a value to the corresponding X.3 parameter. Only certain values are recognized for each specific X.3 parameter. *value1* refers to line-mode processing. *value2* refers to raw-mode processing.

Comment Lines Comment lines begin with a # character, and any characters following the # character are treated as comments (ignored).

X.3 parameters are initially set from the **x3config** file. They can also be set by applications that rely on specific terminal I/O options. X.3 parameters are used to control I/O options between the host and the PAD device, and are analogous to host-to-terminal I/O options. For example, baud rate and parity must be set to appropriate values before a host and terminal can communicate correctly. X.3 parameters provide a means for defining how communication is to be handled between the host and the PAD.

(Requires Optional LAN/X.25 Software)

Here is a summary of the list of X.3 CCITT Parameters:

X.3 Parameter	Description
1	**Escape From Data Transfer**
	0 Not possible
	1 Possible with DLE
	32-126 Possible with defined character
2	**Echo**
	0 No echo 1 Echo
3	**Data Forwarding Signal**
	0 No data forwarding character
	1 Alphanumerics
	2 Carriage return
	4 ESC, BEL, ENQ, ACK
	8 DEL, CAN, DC2
	16 ETX, EOT
	32 HT, LF VT, FF
	64 All other characters
	127 All characters are for data forwarding
4	**Idle Timer**
	0 No Time Out
	1-255 Increments of 1/20 of a second
5	**Ancillary Device Control**
	0 No use of XON and XOFF
	1 Use of XON and XOFF for flow control
6	**Pad Service Signals**
	0 No Service Signals
	1 Service Signals other than prompt
	4 Prompt Service Signal
	5 (1+4)
7	**Procedure on Break**
	0 Nothing
	1 INTERRUPT
	2 RESET
	4 Indication of break pad message
	8 Escape from data transfer
	16 Discard output
	21 (1+4+16)
8	**Discard Output**
	0 Normal Data Delivery
	1 Discard Output
9	**Carriage Return Padding**
	0 No Padding
	1-7 Padding Characters
10	**Line Folding**

(Requires Optional LAN/X.25 Software)

0 No Line Folding
1-255 Number of characters per Line

X.3 Parameter **Description**

11 **Terminal Speed**

Value	Speed	Value	Speed
0	110	9	100
1	134.5	10	50
2	300	11	75/1200
3	1200	12	4800
4	600	14	9600
5	75	15	19200
6	150	16	48000
7	1800	17	56000
8	200	18	64000

12
PAD Flow Control by the Terminal
0 No use of XON and XOFF
1 Flow control by use of XON and XOFF

13
Line-Feed Insertion
0 No LF insertion
1 Insert LF after each CR to the terminal
2 Insert LF after each CR from the terminal
4 Insert LF after each CR sent as echo to the terminal
5 (1+4)
6 (2+4)
7 (1+2+4)

14
Linefeed Padding
0 No LF padding
1-7 Padding characters

15
Editing
0 No editing during data transfer
1 Editing during data transfer

16
Character Delete
0-127 Delete Character

17
Line Delete
0-127 Line-delete Character

18
Line Display
0-127
Line-display character

19
Editing PAD Service Signals
0 No editing PAD service signal
1 Editing PAD service signals for printing terminals
2 Editing PAD service signals for displaying terminals
8, 32-126
Editing PAD service signals using one character from decimal 8 or the range of decimal 32 through 126

20
Echo Mask
- **0** No echo mask at all
- **1** No echo of CR
- **2** No echo of LF
- **4** No echo of VT, HT, FF
- **8** NO echo of BEL, BS
- **16** No echo of ESC, ENQ
- **32** No echo of ACK, NAK, STX, SOH, EOT, ETB, ETX
- **64** No echo of editing characters as designated by Parameter 16, 17, 18
- **128** No echo of all other characters in columns 0 and 1 not mentioned above and DEL
- All possible combination from above in the range 1 through 127 can be used for parameter 20.

21
Parity Treatment
- **0** parity checking or generation
- **1** Parity checking
- **2** Parity generation
- **3** Parity checking and generation

22
Page Wait
- **0** Page-wait disabled
- **1-255** Number of line feed characters considered by the PAD for the page-wait function

EXAMPLES

Here is a sample X.3 configuration file:

```
# Sample x3config file

hp_padsrvr {
# <parameter> <value1> <value2>
      1           1         1
      2           1         1
      3          94       127
      4           0         0
      5           1         1
      6           5         5
      7          21        21
      8           0         0
      9           0         0
     10           0         0
     11          14        14
     12           1         1
     13           0         0
     14           0         0
     15           0         1
     16           8         8
     17          24        24
     18           0         0
     19           1         1
     20           0         0
     21           0         0
     22           0         0
}

hp_profile : 0 {
# <parameter> <value>
      1         1
      2         1
```

(Requires Optional LAN/X.25 Software)

```
                3       127
                4         0
                5         1
                6         5
                7        21
                8         0
                9         0
               10         0
               11        14
               12         1
               13         0
               14         0
               15         1
               16         8
               17        24
               20         0
               21         0
               22         0
        }
```

AUTHOR

`x3config` was developed by HP.

SEE ALSO

x29server(1M), x29printd(1M), x29uucpd(1M), padem(1), x29hosts(4).
Installing and *Administering* X.25/9000.

NAME

ypfiles - Network Information Service database and directory structure

DESCRIPTION

The Network Information Service (NIS) network lookup service uses databases in the directory hierarchy under **/usr/etc/yp**. (Note: a symbolic link exists from **/etc/yp** to **/usr/etc/yp**.) These databases exist only on machines that act as NIS servers. A database consists of two files created by **makedbm** (see *makedbm*(1M)). One has the filename extension **.pag** and the other has the filename extension **.dir**. For example, the database named **netgroup** is implemented by the pair of files **netgroup.pag** and **netgroup.dir**. A database served by the NIS is called an NIS *map*.

An NIS *domain* is a named set of Network Information Service maps. Each NIS domain is implemented as a subdirectory of **/usr/etc/yp** (whose name is the domain name) and contains the maps for that domain. Any number of NIS domains can exist, and each can contain any number of maps.

Besides the databases contained in **/usr/etc/yp/***domain*, master NIS servers have files named *general_NIS_mapname* **.time** that reside there, too. These files are merely empty files whose times of last modification are compared with those of the ASCII files from which the maps are built. The **ypmake** script performs these comparisons to determine whether the maps are current (see *ypmake*(1M)). The *general_NIS_mapname* designation is described further in the FILES section below.

The NIS lookup service does not require maps, although maps may be required for the normal operation of other parts of the system. The list of maps an NIS server provides access to is neither restricted nor must it be all-inclusive. If a map exists in a given domain and a client asks about it, the NIS serves it. For a map to be consistently accessible, it must exist on all NIS servers that serve the domain. To provide data uniformity between the replicated maps, make an entry to run **ypxfr** periodically in root's **crontab** file on each server (see *ypxfr*(1M) and *crontab*(1M)). More information on this topic is in *yppush*(1M) and *ypxfr*(1M).

NIS maps contain two special key-value pairs. The first key, *NIS_LAST_MODIFIED*, has a 10-character (ASCII) order number as a value. The order number is the **time()** in seconds when the map was built (see *time*(2)). The second key is *NIS_MASTER_NAME*, whose value is the host name of the map's master NIS server. The **makedbm** command generates both key-value pairs automatically. The **ypxfr** command uses these values when it transfers a map from one NIS server to another.

Generate and modify NIS maps only on the master server. They are copied to the slaves using **ypxfr** to avoid potential byte-ordering problems among NIS servers running on machines with different architectures, and to minimize the disk space required for the databases (see *ypxfr*(1M)). NIS databases can be created initially for both masters and slaves by using **ypinit** (see *ypinit*(1M)).

After servers' databases are created, the contents of some maps will change. Generally, an ASCII source version of each database exists on the master, and is changed with a text editor. The NIS map is rebuilt to include the changes, and propagated from the master to the slaves by running the **ypmake** shell script (see *ypmake*(1M)).

All standard NIS maps are built by commands contained in the **ypmake** script. If you add a non-standard NIS map, edit this script to support the new map (standard NIS maps are discussed under FILES below). **ypmake** uses **makedbm** to generate the NIS maps on the master and may run **yppush** to copy the rebuilt maps to the slaves (see *yppush*(1M)). The **yppush** command refers to the contents of the map named **ypservers** that contains the host names of all NIS servers for the specific domain. For more information, see *ypmake*(1M), *yppush*(1M), and *ypxfr*(1M).

DEPENDENCIES

If **/usr/etc/yp** is in a file system that does not allow file names longer than 14 characters and you want to create a new non-standard map for the Network Information Service, its name must not exceed 10 characters in length. This rule exists because **makedbm** adds the 4-character suffixes **.dir** and **.pag** to any mapname.

The following table describes the translation of standard NIS mapnames to shorter names for storage on a 14-character filename file system. The standard mapnames should be used by NIS clients on HP machines when making requests, regardless of which machine is the NIS server.

Standard NIS Mapname	Abbreviated Mapname
mail.aliases	mail.alias
ethers.byaddr	ether.byad
ethers.byname	ether.byna
group.bygid	group.bygi
group.byname	group.byna
hosts.byaddr	hosts.byad
hosts.byname	hosts.byna
netgroup	netgroup
netgroup.byhost	netgr.byho
netgroup.byuser	netgr.byus
networks.byaddr	netwk.byad
networks.byname	netwk.byna
passwd.byname	passw.byna
passwd.byuid	passw.byui
protocols.byname	proto.byna
protocols.bynumber	proto.bynu
rpc.bynumber	rpc.bynu
services.byname	servi.byna
vhe_list	vhe_list
ypservers	ypservers

AUTHOR

ypfiles was developed by Sun Microsystems, Inc.

FILES

The following table presents information about the standard Network Information Service maps.

The *General NIS Mapname* column lists names for sets of NIS maps; the sets include adjacent entries from the *Standard NIS Mapname* column.

The *ASCII Source* column lists the ASCII files from which the maps are usually built on HP master NIS servers. The **ypmake** script permits the source directory, or file in the case of the passwd maps, to vary.

The *Standard NIS Mapname* column lists names by which maps are stored on NIS servers and referred to by NIS clients.

General NIS Mapname	ASCII Source	Standard NIS Mapname
aliases	*	mail.aliases
ethers	*	ethers.byaddr
	ethers.byname	
group		/etc/group
		group.byname
hosts		/etc/hosts
		hosts.byname
netgroup	/etc/netgroup	netgroup
		netgroup.byhost
		netgroup.byuser
networks	/etc/networks	networks.byaddr
		networks.byname
passwd		/etc/passwd
		passwd.byuid
protocols	/etc/protocols	protocols.byname
		protocols.bynumber
rpc	/etc/rpc	rpc.bynumber
services	/etc/services	services.byname
vhe_list	/etc/vhe_list	vhe_list **
ypservers	***	ypservers

* These databases are not built on HP master Network Information Service servers. However, if an HP machine is a slave to a master NIS server that creates and distributes these databases, the HP slave NIS server will store these databases. It is suggested that if you have a non-HP machine that requires these maps, make that machine the master NIS server. By doing this, the maps should be built as needed.

** The **vhe_list** map is a map generated only by HP master NIS servers.

*** No ASCII source exists for the **ypservers** database. It is created from responses provided by the user of **ypinit** on the master NIS server, and it has no matching **ypservers.time** file.

SEE ALSO

domainname(1M), makedbm(1M), rpcinfo(1M), vhe_altlog(1M), vhe_mounter(1M), vhe_u_mnt(1M), ypinit(1M), ypmake(1M), yppoll(1M), yppush(1M), ypserv(1M), ypxfr(1M), vhe_list(4).

Section 5:
Miscellaneous Topics

NAME

intro - introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, and the file system hierarchy.

SEE ALSO

The introduction to this manual.

NAME

acl - introduction to access control lists

DESCRIPTION

Access control lists are a key enforcement mechanism of discretionary access control (see Definitions below), for specifying access to files by users and groups more selectively than traditional HP-UX mechanisms allow.

HP-UX already enables non-privileged users or processes, such as file owners, to allow or deny other users access to files and other objects on a "need to know" basis, as determined by their user and/or group identity (see *passwd*(4) and *group*(4)). This level of control is accomplished by setting or manipulating a file's permission bits to grant or restrict access by owner, group, and others (see *chmod*(2)).

ACLs offer a greater degree of selectivity than permission bits. ACLs allow the file owner or superuser to permit or deny access to a list of users, groups, or combinations thereof.

ACLs are supported as a superset of the UNIX operating system discretionary access control (DAC) mechanism for files, but not for other objects such as inter-process communication (IPC) objects.

Definitions

Because control of access to data is a key concern of computer security, we provide the following definitions, based on those of the *Department of Defense Trusted Computer System Evaluation Criteria,* to explain further both the concepts of access control and its relevance to HP-UX security features:

access "A specific type of interaction between a subject and an object that results in the flow of information from one to the other." Subjects include "persons, processes, or devices that cause information to flow among objects or change the system state." Objects include files (ordinary files, directories, special files, FIFOs, etc.) and inter-process communication (IPC) features (shared memory, message queues, semaphores, sockets).

access control list (ACL)
An access control list is a set of (*user.group, mode*) entries associated with a file that specify permissions for all possible user-ID/group-ID combinations.

access control list (ACL) entry
An entry in an ACL that specifies access rights for one user and group combination.

change permission
The right to alter DAC information (permission bits or ACL entries). Change permission is granted to object (file) owners and to privileged users.

discretionary access control (DAC)
"A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) to any other subject."

mode Three bits in each ACL entry which represent read, write, and execute/search permissions. These bits may exist in addition to the 16 mode bits associated with every file in the file system (see *glossary*(9)).

privilege The ability to ignore access restrictions and change restrictions imposed by security policy and implemented in an access control mechanism. In HP-UX, superusers and members of certain groups (see *privgrp*(4)) are the only privileged users.

restrictive versus permissive
An individual ACL entry is considered restrictive or permissive, depending on context. Restrictive entries deny a user and/or group access that would otherwise be granted by less-specific base or optional ACL entries (see below). Permissive entries grant a user and/or group access that would otherwise be denied by less-specific base or optional ACL entries.

Access Control List Entries

An access control list (ACL) consists of sets of (*user.group, mode*) entries associated with a file that specify permissions. Each entry specifies for one user-ID/group-ID combination a set of access permissions, including read, write, and execute/search.

To help understand the relationship between access control lists and traditional file permissions, consider the following file and its permissions:

```
-rwxr-xr--    james  admin  datafile
```

The file owner is user **james**.
The file's group is **admin**.
The name of the file is **datafile**.
The file owner permissions are **rwx**.
The file group permissions are **r-x**.
The file other permissions are **r--**.

In an ACL, user and group IDs can be represented by names or numbers, found in **/etc/passwd**. The following special symbols can also be used:

%	Symbol representing no specific user or group.
@	Symbol representing the current file owner or group.

Base ACL Entries

When a file is created, three base access control list entries are mapped from the file's access permission bits to match a file's owner and group and its traditional permission bits. Base ACL entries can be changed by the *chmod*(2) and *setacl*(2) system calls.

(*uid*.%,mode)	Base ACL entry for the file's owner
(%.*gid*,mode)	Base ACL entry for the file's group
(%.%,mode)	Base entry for other users

(Except where noted, examples are represented in short form notation. See ACL Notation, below.)

Optional ACL entries

Optional access control list entries contain additional access control information, which the user can set with the *setacl*(2) system call to further allow or deny file access. Up to thirteen additional user/group combinations can be specified.

For example, the following optional access control list entries can be associated with our file:

(mary.admin, rwx)	Grant read, write, and execute access to user **mary** in group **admin**.
(george.%, ---)	Deny any access to user **george** in no specific group.

ACL Notation

Supported library calls and commands that manage ACLs recognize three different symbolic representations:

operator form	For input of entire ACLs and modifications to existing ACLs, in a syntax similar to that used by *chmod*(1).
short form	Easier to read, intended primarily for output. *Chacl*(1) accepts this form as input so that it can interpret output from *lsacl*(1).
long form	A multi-line format useful for greater clarity, and supported only for output.

For our example file, the base ACL entries could be represented in the three notations as follows:

operator form	james.% = rwx, %.admin = rx, %.% = r
short form	(james.%,rwx) (%.admin,r-x) (%.%,r--)
long form	rwx james.% r-x %.admin r-- %.%

In addition to basic ACL usage, some library calls and commands understand and use a variation of operator and short forms. See the section below on *ACL Patterns*.

ACL Uniqueness

Entries are unique in each ACL. There can only be one (*u.g, mode*) entry for any pair of *u* and *g* values; one (*u.%, mode*) entry for a given value of *u*; one (*%.g, mode*) entry for a given value of *g*; and one (*%.%, mode*) entry for each file. For example, an ACL can have a (23.14, *mode*) entry and a (23.%, *mode*) entry, but not two (23.14, *mode*) entries or two (23.%, *mode*) entries.

Access Check Algorithm

ACL entries can be categorized by four levels of specificity. In access checking, ACLs are compared to the effective user and group IDs in this order:

(*u.g*, rwx)	specific user, specific group
(*u.%*, rwx)	specific user, no specific group
(*%.g*, rwx)	no specific user, specific group
(*%.%*, rwx)	no specific user, no specific group

Once an entry for the combination of a process effective user ID and effective group ID (or any supplementary group ID) is matched, no further (that is, less specific) entries are checked. More specific entries that match take precedence over any less specific ones that also match.

If a process has more than one group ID (that is, a non-null supplementary groups list), more than one (*u.g, mode*) or (*%.g, mode*) entry might apply for that process. If so, the access modes in all matching entries (of the same level of specificity, *u.g* or *%.g*) are OR'd together. Access is granted if the resulting mode bits allow it. Since entries are unique, the order of entries in each entry type is insignificant.

Because the traditional UNIX permission bits are mapped into base ACL entries, they are included in access checks.

If a request is made for more than one type of access, such as opening a file for both reading and writing, access is granted only if the process is allowed all requested types of access. Note that access can be granted if the process has two groups in its groups list, one of which is only allowed read access, and the other of which is only allowed write access. In other words, even if the requested access is not granted by any one entry, it may be granted by a combination of entries due to the process belonging to several groups.

Operator Form of ACLs (input only)

user . *group operator mode* [*operator mode*]... , ...

Multiple entries are separated by commas, as in *chmod*(1). Each entry consists of a user identifier and group identifier followed by one or more operators and mode characters, as in the mode syntax accepted by *chmod*(1).

The entire ACL must be a single argument, and thus should be quoted to the shell if it contains whitespace or special characters. Whitespace is ignored except within names. A null ACL is legitimate, and means either "no access" or "no changes", depending on context.

Each user or group ID may be represented by:

name	Valid user or group name.
number	Valid numeric ID value.
%	"No specific user or group," as appropriate.
@	"Current file owner or group," as appropriate; useful for referring to a file's *u.%* and *%.g* base ACL entries.

An operator is always required in each entry. Operators are:

- **=** Set all bits in the entry to the given mode value.
- **+** Set the indicated mode bits in the entry.
- **-** Clear the indicated mode bits in the entry.

The mode is represented by an octal value of **0** through **7**; or any combination of **r**, **w**, and **x** can be given in any order (see EXAMPLES below). A null mode denies access if the operator is **=**, or represents "no change" if the operator is **+** or **-**.

Multiple entries and multiple operator-mode parts in an entry are applied in the order specified. Conflicts do not result in error; the last specified entry or operator takes effect. Entries need not appear in any particular order.

Note that *chmod*(1) allows only **u**, **g**, **o**, or **a** to refer symbolically to the file owner, group, other, or all users, respectively. Since ACLs work with arbitrary user and group identifiers, **@** is provided as a convenience.

The exact syntax is:

```
acl     ::= [entry[,entry]...]
entry   ::= id . id op mode [op mode]...
id      ::= name | number | % | @
op      ::= = | + | -
mode    ::= 0..7 | [char[char]...]
char    ::= r | w | x
```

Short Form of ACLs (input and output)
(user . group, mode) ...

Short form differs from operator form in several ways:

- Entries are surrounded by parentheses rather than being separated by commas.
- Each entry specifies the mode, including all mode bits. It is not possible to change the mode value with **+** and **-** operators. However, the comma functions like the **=** operator in operator form.
- For clarity, hyphens represent unset permission bits in the output of the mode field and are allowed in input. This resembles the mode output style used by *ls*(1).

Multiple entries are concatenated. For consistency with operator form, a dot (.) is used to separate user and group IDs.

On output, no whitespace is printed except in names (if any). ID numbers are printed if no matching names are known. Either ID can be printed as **%** for "no specific user or group." The mode is represented as <**r** | -><**w** | -><**x** | ->, that is, it always has three characters, padded with hyphens for unset mode bits. If the ACL is read from the system, entries are ordered by specificity, then by numeric values of ID parts.

On input, the entire ACL must be a single argument, and thus should be quoted to the shell if it contains whitespace or special characters. Whitespace is ignored except within names. A null ACL is legitimate, and means either "no access" or "no changes", depending on context.

User and group IDs are represented as in operator form.

The mode is represented by an octal value of **0** through **7**; or any combination of **r**, **w**, **x** and **-** (ignored) can be given in any order (see EXAMPLES below). A null mode denies access.

Redundancy does not result in error; the last entry for any user-ID/group-ID combination takes effect. Entries need not appear in any particular order.

The exact syntax is:

```
acl     ::= [entry[entry]...]
entry   ::= (id.id,mode)
id      ::= name | number | % | @
mode    ::= 0..7 | [char[char]...]
char    ::= r | w | x | -
```

Long Form of ACLs (output only)
mode user . group

Each entry occupies a single line of output. The mode appears first in a fixed-width field, using hyphens (for unset mode bits) for easy vertical scanning. Each user and group ID is shown as a name if known, a number if unknown, or **%** for "no specific user or group." Entries are ordered from most to least specific, then by numeric values of ID parts.

Note that every ACL printed has at least three entries, the base ACL entries (that is, *uid.%*, *%.gid*, and *%.%*).

The exact syntax is:

```
acl     ::= entry[<newline>entry]...
entry   ::= mode<space>id.id
mode    ::= <r | -><w | -><x | ->
```

id ::= *name* | *number* | %

ACL Patterns

Some library calls and commands recognize and use ACL patterns instead of exact ACLs to allow operations on all entries that match the patterns. ACL syntax is extended in the following ways:

wildcard user and group IDs
: A user or group name of * (wildcard) matches the user or group ID in any entry, including % (no specific user or group).

mode bits on, off, or ignored
: For operator-form input, the operators =, +, and - are applied as follows:

  = entry mode value matches this mode value exactly
  + these bits turned on in entry mode value
  - these bits turned off in entry mode value

  When only + and - operators are used, commands ignore the values of unspecified mode bits.

  Short-form patterns treat the mode identically to the = operator in operator form.

wildcard mode values
: A mode of * (wildcard) in operator or short form input (for example, "ajs.%=*" or "(ajs.%,*)") matches any mode value, provided no other mode value is given in a operator-form entry. Also, the mode part of an entry can be omitted altogether for the same effect.

entries not combined
: Entries with matching user and group ID values are not combined. Each entry specified is applied separately by commands that accept patterns.

ACL Operations Supported

The system calls *setacl*(2) and *getacl*(2) allow setting or getting the entire ACL for a file in the form of an array of *acl_entry* structures. To check access rights to a file, see *access*(2) and *getaccess*(2).

Various library calls are provided to manage ACLs:

acltostr(3C) Convert *acl_entry* arrays to printable strings.

strtoacl(3C) Parse and convert ACL strings to *acl_entry* arrays.

strtoaclpatt(3C)
: Parse and convert ACL pattern strings to *acl_entry_patt* arrays.

setaclentry(3C)
fsetaclentry Add, modify, or delete a single ACL entry in one file's ACL.

cpacl(3C)
fcpacl Copy an ACL and file miscellaneous mode bits (see *chmod*(2)) from one file to another, transfer ownership if needed (see below), and handle remote files correctly.

chownacl(3C) Change the file owner and/or group represented in an ACL, that is, transfer ownership (see below).

The following commands are available to manage ACLs and permissions:

chacl(1) Add, modify, or delete individual entries or all optional entries in ACLs on one or more files, remove all access to files, or incorporate ACLs into permission bits.

lsacl(1) List ACLs on files.

chmod(1) Change permission bits and other file miscellaneous mode bits.

ls(1) In long form, list permission bits and other file attributes.

find(1) Find files according to their attributes, including ACLs.

getaccess(1) List access rights to file(s).

ACL Interaction with *stat*(2), *chmod*(2), and *chown*(2)

stat The *st_mode* field summarizes the caller's access rights to the file. It differs from file permission bits only if the file has one or more optional entries applicable to the caller. The *st_basemode* field provides the file's actual permission bits. The *st_acl* field indicates the presence of optional ACL entries in the file's ACL.

The *st_mode* field contains a user-dependent summary, so that programs ignorant of ACLs that use *stat*(2) and *chmod*(2) are more likely to produce expected results, and so that *stat*(2) provides reasonable information about remote files over NFS. The *st_basemode* and *st_acl* fields are useful only for local files.

chmod For conformance with IEEE Standard POSIX 1003.1-1988, *chmod*(2) deletes any optional entries in a file's ACL. Unfortunately, since *chmod*(2) is used to set file miscellaneous mode bits as well as permission bits, extra effort is required in some cases to preserve a file's ACL.

chown If the new owner and/or group of a file does not already have an optional (*u.%, mode*) and/or (*%.g, mode*) entry in the file's ACL, it inherits the old owner's and/or group's file access permission bits and base ACL entry:

(*id1,mode1*) -> (*id2,mode1*)

This is the traditional behavior. However, if the new owner and/or group of a file already has an optional (*u.%, mode*) and/or (*%.g, mode*) entry in the file's ACL, the ACL does not change:

(*id1, mode1*) -> (*id1, mode1*)
(*id2, mode2*) -> (*id2, mode2*)

Existing access information in the ACL is preserved. However, because the old optional ACL entry becomes the new base ACL entry and vice versa, the file's access permission bits change.

Transferring ownership of ACLs by *chown*(2) allows a file to be transferred to a different user or group, or copied by a different user or group than the owner (using *cpacl*(3C) or *chownacl*(3C)), and later returned to the original owner or group without net changes to its ACL. The extra complexity is necessary because:

- ACLs are a backward-compatible superset of permission bits (which are coupled to file owner and group IDs), not a replacement for them.
- it enables users and programs that deal with ACLs to do so simply, rather than with a combination of permission bits and ACL entries. Also, the access check algorithm is simpler and more symmetrical; permission bits do not "eclipse" or "mask" ACL entries.

EXAMPLES

Operator Form

The following sets the *%.%* entry to restrict "other" users to only reading the file.

chacl '%.% = r' myfile

The following allows user "bill" in any group to write the file, assuming that no restrictive entry is more specific than the **bill.%** entry (for example, a **bill.adm** entry that denies writing).

chacl 'bill.% +w' myfile

The following ACL specification contains two entries. The first one deletes write and adds read capability to the entry for user 12, group 4. The second entry denies access for any unspecified user in any unspecified group.

chacl '12.4-w+r, %.% =' myfile

The following pair of entries sets the *u.%* entry for the file's owner to allow both read and execute and results in adding write and execute capabilities for "other" users (the "*%.%*" entry). Note that a mode character is purposely repeated for illustration purposes.

chacl '@.% = 5, %.% + xwx' myfile

Short Form

Here is a typical ACL as it might be printed. It allows user **jpc** to read or execute the file while in group **adm**; it denies user **ajs** access to the file while in group **trux**; it allows user **jpc** in any group (except **adm**) to only read the file; any other user in group **bin** may read or execute the file; and any other user may only read the file.

(jpc.adm,r-x)(ajs.trux,---)(jpc.%,r--)(%.bin,r-x)(%.%,r--)

The following allows "other" users to only read the file.

chacl '(%.%,r)' myfile

The following sets write-only access for user **bill** in any group.

chacl '(bill.%,-w-)' myfile

The following sets the entry for user 12 in group 4 to allow read and write.

chacl '(12.4,wr)' myfile

The following sets the base ACL entry for the file's owner to allow both read and execute, and sets write and execute capabilities for "other" users (the "%.%" entry).

chacl '(@.%, 5) (%.%, xwx)' myfile

Long Form

Here is the same ACL as in an earlier example, printed in long form.

```
r-x    jpc.adm
---    ajs.trux
r--    jpc.%
r-x    %.bin
r--    %.%
```

ACL Patterns

The following command locates files whose ACLs contain an entry that allows read access and denies write access to some user/group combination.

find / -acl '*.*+r-w' -print

The following matches entries for any user in group **bin** and for user **tammy** in any group, regardless of the entries's mode values. Matching optional ACL entries are deleted and mode values in matching base ACL entries are set to zero:

chacl -d '%.bin, tammy.*=*' myfile

The following matches all entries, deleting optional entries and setting mode values of base ACL entries to zero:

chacl -d '(*.*,*)' myfile

HEADERS

Header <sys/acl.h>

The **<sys/acl.h>** header file defines the following constants to govern the numbers of entries per ACL:

NACLENTRIES	maximum number of entries per ACL, including base entries
NBASEENTRIES	number of base entries
NOPTENTRIES	number of optional entries

The ACL entry structure **struct acl_entry** is also defined, and includes the following members:

```
aclid_t     uid;
aclid_t     gid;
aclmode_t   mode;
```

The **<sys/acl.h>** header also defines the types **aclid_t** and **aclmode_t.**

Non-specific user and group ID values:

ACL_NSUSER	non-specific user ID
ACL_NSGROUP	non-specific group ID

A special *nentries* value ACL_DELOPT is used with *setacl*(2) to delete optional entries.

Header <sys/getaccess.h>

The **<sys/getaccess.h>** header defines constants for use with *getaccess*(2).

Special parameter values for *uid*:

UID_EUID use effective user ID
UID_RUID use real user ID
UID_SUID use saved user ID

Special parameter values for *ngroups*:

NGROUPS_EGID process's effective gid
NGROUPS_RGID process's real gid
NGROUPS_SGID process's saved gid
NGROUPS_SUPP process's supplementary groups only
NGROUPS_EGID_SUPP process's eff gid plus supp groups
NGROUPS_RGID_SUPP process's real gid plus supp groups
NGROUPS_SGID_SUPP process's saved gid plus supp groups

Header <acllib.h>

The **<acllib.h>** header file defines several constants for use with ACL support library calls.

Symbolic forms of ACLs for *acltostr()*:

FORM_SHORT
FORM_LONG

Magic values for various calls:

ACL_FILEOWNER file's owner ID
ACL_FILEGROUP file's group ID
ACL_ANYUSER wildcard user ID
ACL_ANYGROUP wildcard group ID
MODE_DEL delete one ACL entry

Mask for valid mode bits in ACL entries:

MODEMASK (R_OK | W_OK | X_OK)

The **<acllib.h>** header also defines the **struct acl_entry_patt** ACL pattern entry structure, which includes the following members:

```
aclid_t     uid;     /* user ID */
aclid_t     gid;     /* group ID */
aclmode_t   onmode;  /* mode bits that must be on */
aclmode_t   offmode; /* mode bits that must be off */
```

WARNINGS

ACLs are intended for use on ordinary files and directories. Optional ACL entries are not recommended on files that are manipulated by certain system utilities, such as terminal special files and LP scheduler control files. These utilities might delete optional entries, including those whose intent is restrictive, without warning as a consequence of calling *chmod*(2), thereby increasing access unexpectedly.

Most, but not all, supported utilities are able to handle ACLs correctly. However, only the *fbackup*(1M) and *frecover*(1M) file archive utilities handle access control lists properly. When using programs (such as archive programs *ar*(1), *cpio*(1), *ftio*(1), *tar*(1), and *dump*(1M)) unable to handle ACLs on files with optional ACL entries, note the Access Control List information included on their respective reference pages, to avoid loss of data.

If a user name is defined in the **/etc/passwd** file or a group name is defined in the **/etc/group** file as % or @, or for patterns, *, ACL syntax cannot reference that name as itself because the symbols have other meanings. However, such users or groups can still be referenced by their ID numbers. User and/or group names must not include the following characters:

. Do not use in user names.
+ Do not use in group names.
- Do not use in group names.
= Do not use for operator form input of group names.
, Do not use for short form or for operator form patterns.
) Do not use for short form patterns.

It is possible to specify an ACL pattern using the @ (file owner or group) or * (wildcard) symbols so that it cannot match certain files, perhaps depending on their ownership, by giving two entries, one with specific

values and the other using @ or *, which are equivalent for a file but contain different mode values. For example:

find / -acl '(ajs.%,r)(@.%,rw)' -print

cannot match a file owned by **ajs**.

DEPENDENCIES

NFS NFS. does not support ACLs on remote files. Individual manual entries specify the behavior of various system calls, library calls, and commands under these circumstances. Be careful when transferring a file with optional entries over a network or when manipulating a remote file because optional entries may be silently deleted.

AUTHOR

The access control list design described here was developed by HP.

FILES

< sys/acl.h >	Header file that supports *setacl*(2) and *getacl*(2).
< sys/getaccess.h >	Header file that supports *getaccess*(2).
< acllib.h >	Header file that supports ACL library calls.
/etc/passwd	Defines user names and user and group ID values.
/etc/group	Defines group names.

SEE ALSO

chacl(1), chmod(1), cp(1), find(1), getaccess(1), ln(1), ls(1), lsacl(1), mv(1), rm(1), fbackup(1M), frecover(1M), fsck(1M), fsdb(1M) access(2), chmod(2), chown(2), creat(2), getaccess(2), getacl(2), mknod(2), open(2), setacl(2), stat(2), acltostr(3C), chownacl(3C), cpacl(3C), setaclentry(3C), strtoacl(3C), group(4), passwd(4), privgrp(4).

NAME

ascii - map of ASCII character set

SYNOPSIS

```
cat /usr/pub/ascii
```

DESCRIPTION

`/usr/pub/ascii` provides a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. The file contains the following text:

```
|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel|
|010 bs |011 ht |012 nl |013 vt |014 np |015 cr |016 so |017 si |
|020 dle|021 dc1|022 dc2|023 dc3|024 dc4|025 nak|026 syn|027 etb|
|030 can|031 em |032 sub|033 esc|034 fs |035 gs |036 rs |037 us |
|040 sp |041  ! |042  " |043  # |044  $ |045  % |046  & |047  ´ |
|050  ( |051  ) |052  * |053  + |054  , |055  - |056  . |057  / |
|060  0 |061  1 |062  2 |063  3 |064  4 |065  5 |066  6 |067  7 |
|070  8 |071  9 |072  : |073  ; |074  < |075  = |076  > |077  ? |
|100  @ |101  A |102  B |103  C |104  D |105  E |106  F |107  G |
|110  H |111  I |112  J |113  K |114  L |115  M |116  N |117  O |
|120  P |121  Q |122  R |123  S |124  T |125  U |126  V |127  W |
|130  X |131  Y |132  Z |133  [ |134  \ |135  ] |136  ^ |137  _ |
|140  ` |141  a |142  b |143  c |144  d |145  e |146  f |147  g |
|150  h |151  i |152  j |153  k |154  l |155  m |156  n |157  o |
|160  p |161  q |162  r |163  s |164  t |165  u |166  v |167  w |
|170  x |171  y |172  z |173  { |174  | |175  } |176  ~ |177 del|

| 00 nul| 01 soh| 02 stx| 03 etx| 04 eot| 05 enq| 06 ack| 07 bel|
| 08 bs | 09 ht | 0a nl | 0b vt | 0c np | 0d cr | 0e so | 0f si |
| 10 dle| 11 dc1| 12 dc2| 13 dc3| 14 dc4| 15 nak| 16 syn| 17 etb|
| 18 can| 19 em | 1a sub| 1b esc| 1c fs | 1d gs | 1e rs | 1f us |
| 20 sp | 21  ! | 22  " | 23  # | 24  $ | 25  % | 26  & | 27  ´ |
| 28  ( | 29  ) | 2a  * | 2b  + | 2c  , | 2d  - | 2e  . | 2f  / |
| 30  0 | 31  1 | 32  2 | 33  3 | 34  4 | 35  5 | 36  6 | 37  7 |
| 38  8 | 39  9 | 3a  : | 3b  ; | 3c  < | 3d  = | 3e  > | 3f  ? |
| 40  @ | 41  A | 42  B | 43  C | 44  D | 45  E | 46  F | 47  G |
| 48  H | 49  I | 4a  J | 4b  K | 4c  L | 4d  M | 4e  N | 4f  O |
| 50  P | 51  Q | 52  R | 53  S | 54  T | 55  U | 56  V | 57  W |
| 58  X | 59  Y | 5a  Z | 5b  [ | 5c  \ | 5d  ] | 5e  ^ | 5f  _ |
| 60  ` | 61  a | 62  b | 63  c | 64  d | 65  e | 66  f | 67  g |
| 68  h | 69  i | 6a  j | 6b  k | 6c  l | 6d  m | 6e  n | 6f  o |
| 70  p | 71  q | 72  r | 73  s | 74  t | 75  u | 76  v | 77  w |
| 78  x | 79  y | 7a  z | 7b  { | 7c  | | 7d  } | 7e  ~ | 7f del|
```

FILES

/usr/pub/ascii

NAME

Audio - audio application interface and demo program

Remarks:

Beginning with Release 8.07, the HP-UX operating system includes audio software comprised of an Audio Application Program Interface (AAPI) and some example programs.

The audio software package contains client and server components, which can run on separate systems. Audio data may reside on still a third system. In all cases, however, the server must run on a Series 700 system equipped with audio hardware (to determine the presence of audio hardware, check the hardware manual provided with the system or look for audio jacks on the back of the computer enclosure).

DESCRIPTION

Audio Applications Programming Interface

The Audio Applications Programming Interface (AAPI) includes a library of functions that can be called by an application program written in C language. The functions interact with the audio server, enabling the application to record and play audio data files and also convert audio data files from one format to another.

The AAPI also includes audio widgets for play and record, and a toolkit of functions that initialize, register, and unregister these widgets. The toolkit and widgets enable application programs based on Motif or similar graphical user interface toolkits to integrate audio capabilities.

For more information about programming with the AAPI, refer to the manual *Using the Audio Application Program Interface* included in the HP-UX General Programming manual set.

Audio Demonstration Program

The audio demonstration program is designed to demonstrate many of the capabilities of the AAPI. It provides a Motif-like interface to play, record, and edit functionality. A waveform is displayed to facilitate editing and traversal of the audio selection.

After connecting a microphone to the input jack on your system, you can use the demonstration program to create and record an audio file. To ensure user privacy, be sure to turn off the microphone when it is not in use.

To use the play features of the demonstration program, one or more audio data files must be present. These can be created by using the program's record feature, or obtained from another source. Supported file formats are listed in the next section.

You can open an audio file, play it, look at its waveform, and use the waveform to edit the file. To send output to a speaker or headphone connected to the output jack on your system, direct the play output to the external device.

To run the demonstration program, follow these steps:

Step 1. Start the NCS Local Location Broker Daemon.

Task 1. If you are not already superuser, log in as superuser.

Task 2. Enter the command line `/usr/etc/ncs/llbd&`. To make `llbd` start automatically at boot time, edit the file `/etc/netncsrc` and change the line `START_LLBD=0` to `START_LLBD=1`.

Step 2. Reboot.

Step 3. Set the `AUDIO` environment variable to specify the node where the audio client should look for the audio server. If `AUDIO` is not set or if it is set to `:0`, the client connects with a server on the same node.

To specify that the client should connect to a server on another node, set

```
AUDIO = node_name: (Korn, Bourne, and POSIX shells)
export AUDIO
```

or

```
setenv AUDIO node_name: (C shell)
```

Step 4. Normally, the audio server starts whenever the system is booted. Check for the existence of the `Aserver` processes by typing

```
ps -e | grep Aserver
```

You should see *two* **Aserver** processes. If the server is not running, start the audio server by hand by typing

```
/usr/audio/bin/Aserver
```

Then type

```
ps -e | grep Aserver
```

and check that there are two active server processes.

Step 5.
Start the demonstration program by typing **/usr/audio/bin/audio_demo**.

There is online help for the demonstration program. **Help** is a selection on the pulldown menus, and there is a help button at the upper-right-hand corner of the demonstration window.

Audio File Types

Audio data files exist in a variety of formats. The AAPI supports optional extensions to the file name to indicate the sampling rate and the file type, using the format: *filename.sampling_rate.file_type* where *file_type* can be any of the following supported types:

.u	Mulaw
.al	Alaw
.au	Sun (NeXT)
.wav	RIFF (MicroSoft RIFF Waveform)
.snd	NeXT
.116	Linear16(16-bit signed)
.18	Linear8(8-bit signed)
.1o8	Linear8Offset(8-bit unsigned)

The *sampling_rate* extension requires values in the form *n*, *n***k**, or *n***K** to indicate the number of samples per second. Typical sampling rates range from **8k** to **22k**. You can add a *file_type* extension without including a *sampling_rate* extension, and you do not have to mark a *sampling_rate* placeholder with an extra period.

If you have a "Mac" file, try treating it as a raw data file in Linear8Offset with a sampling rate of 22K or another sampling rate.

Adding a Drag and Drop Zone

An audio drag and drop zone can be added. Refer to *Using the Audio Application Program Interface* manual for instructions.

Sample Sound Files

Sample sound files are installed in **/usr/audio/examples**.

DEPENDENCIES

When an application program or the audio demonstration program uses the AAPI, the AAPI audio server component must run on a system that has audio hardware. Note that HP-UX for an 8-Mbyte HP 9000 Model 705 system does not include audio software.

AUTHOR

The AAPI and the audio demonstration program were developed by HP.

SEE ALSO

AAudioString(3X), ABestAudioAttributes(3X), ACheckEvent(3X), ACheckMaskEvent(3X), ACloseAudio(3X), AConnectionNumber(3X), AConnectRecordStream(3X), AConvertAFile(3X), ACreateSBucket(3X), ADataFormats(3X), ADestroySBucket(3X), AEventsQueued(3X), AGetErrorText(3X), AGetChannelGain(3X), AGetGain(3X), AGetSBucketData(3X), AGetSystemChannelGain(3X), AGetTransStatus(3X), AGMGainRestricted(3X), AGrabServer(3X), AInputChannels(3X), AInputSources(3X), ALoadAFile(3X), AMaskEvent(3X), AMaxInputGain(3X), AMaxOutputGain(3X), AMinInputGain(3X), AMinOutputGain(3X), ANextEvent(3X), ANumDataFormats(3X), ANumSamplingRates(3X), AOpenAudio(3X), AOutputChannels(3X), AOutputDestinations(3X),

APauseAudio(3X), APeekEvent(3X), APlaySBucket(3X), APlaySStream(3X), AProtocolRevision(3X), AProtocolVersion(3X), APutBackEvent(3X), APutSBucketData(3X), AQLength(3X), AQueryAFile(3X), ARecordADData(3X), ARecordSStream(3X), AResumeAudio(3X), ASamplingRates(3X), ASaveSBucket(3X), ASelectInput(3X), Aserver(1M), AServerVendor(3X), ASetChannelGain(3X), ASetCloseDownMode(3X), ASetErrorHandler(3X), ASetGain(3X), ASetIOErrorHandler(3X), ASetSystemChannelGain(3X), ASetSystemPlayGain(3X), ASetSystemRecordGain(3X), ASimplePlayer(3X), ASimpleRecorder(3X), ASoundBitOrder(3X), ASoundByteOrder(3X), AStopAudio(3X), AUngrabServer(3X), AVendorRelease(3X), AtAddCallback(3X), AtInitialize(3X), AtRemoveCallback(3X), AuCreatePlay(3X), AuCreateRecord(3X), AuInvokePlay(3X), AuInvokeRecord(3X), AuPlayWidget(3X), AuRecordWidget(3X), AuSaveFile(3X).

Using the Audio Application Program Interface,
Audio Users Guide.

NAME

audit - introduction to HP-UX Auditing System

SYNOPSIS

#include <sys/audit.h>

DESCRIPTION

The purpose of the auditing system is to record instances of access by subjects to objects and to allow detection of any (repeated) attempts to bypass the protection mechanism and any misuses of privileges, thus acting as a deterrant against system abuses and exposing potential security weaknesses in the system.

User and Event Selection

The auditing system provides administrators with a mechanism to select users and activities to be audited. Users are assigned unique identifiers called **audit ids** by the administrator which remain unchanged throughout a user's history. The *audusr*(1M) command is used to specify those users who are to be audited. The *audevent*(1M) command is used to specify system activities (auditable events) that are to be audited. Auditable events are classified into several categories, illustrated by the event category list at the end. (An event category consists of a set of operations that affect a particular aspect of the system.)

Self-auditing Programs

To reduce the amount of log data and to provide a higher-level recording of some typical system operations, a collection of privileged programs are given capabilities to perform self-auditing. This means that the programs can suspend the currently specified auditing on themselves and produce a high-level description of the operations they perform. These self-auditing programs include: *at*(1), *chfn*(1), *chsh*(1), *crontab*(1), *login*(1), *newgrp*(1), *passwd*(1), *audevent*(1M), *audisp*(1M), *audsys*(1M), *audusr*(1M), *cron*(1M), *init*(1M), *lpsched*(1M), *pwck*(1M), and *sam*(1M). Note that only these privileged programs are allowed to do self-auditing, and that the audit suspension they perform only affects these programs and does not affect any other processes on the system.

Viewing of Audited Data

The *audisp*(1M) command is used to view audited data recorded in log file(s). *audisp*(1M) merges the log file(s) into a single audit trail in chronological sequence. The administrator can select viewing criteria provided by *audisp*(1M) to limit the search to particular kinds of events which the administrator is interested in investigating.

Monitoring the Auditing System

To ensure that the auditing system operates normally and that any abnormal behaviors are detected, a privileged *daemon* program, *audomon*(1M), runs in the background to monitor various auditing system parameters. When these parameters take on abnormal (dangerous) values, or when components of the auditing system are accidentally removed, *audomon*(1M) prints warning messages and tries to resolve the problem if possible.

Starting and Halting the Auditing System

The administrator can use the *audsys*(1M) command to start or halt the auditing system, or to get a brief summary of the status of the audit system. Prior to starting the auditing system, *audsys*(1M) also validates the parameters specified, and ensures that the auditing system is in a safe and consistent state.

Audit Log Files

At any time when the auditing system is enabled, at least an audit log file must be present, and another back-up log file is highly recommended. Both of these files (along with various attributes for these files) can be specified using *audsys*(1M). When the current log file exceeds a pre-specified size, or when the auditing file system is dangerously full, the system automatically switches to the back-up file if possible. If a back-up log file is not available, warning messages are sent to request appropriate administrator action.

Event Categories

create Log all creations of objects (files, directories, other file objects), including *creat*(2), *mknod*(2), *pipe*(2), *mkdir*(2), *semget*(2), *msgget*(2), *shmget*(2), and *shmat*(2).

delete Log all deletions of objects (files, directories, other file objects), including *rmdir*(2), *semctl*(2), and *msgctl*(2).

moddac Log all modifications of object's DAC (*chmod*, *setacl*), including *chmod*(2), *chown*(2), *umask*(2), *fchown*(2), *fchmod*(2), *setacl*(2), and *fsetacl*(2).

modaccess Log all modifications other than DAC, including *link*(2), *unlink*(2), *chdir*(2), *setuid*(2), *setgid*(2), *chroot*(2), *setgroups*(2), *setresuid*(2), *setresgid*(2), *rename*(2), *shmctl*(2), *shmdt*(2), and *newgrp*(1).

open Log all openings of objects (file open, other objects open) including *open*(2), *execv*(2), *ptrace*(2), *execve*(2), truncate (2), *ftruncate*(2), and *lpsched*(1M).

close Log all closings of objects (file close, other objects close) including *close*(2).

process Log all operations on processes, including *exit*(2), *fork*(2), *vfork*(2), and *kill*(2).

removable Log all removable media events (mounting and unmounting events), including *smount*(2), *umount*(2), and *vfsmount*(2).

login Log all logins and logouts, including *login*(1), *init*(1M).

admin Log all administrative and privileged events, including *stime*(2), *cluster*(2), *swapon*(2), *settimeofday*(2), *sethostid*(2), *privgrp*(2), *setevent*(2), *setaudproc*(2), *audswitch*(2), *setaudid*(2), *setdomainname*(2), *reboot*(2), *sam*(1M), *audisp*(1M), *audevent*(1M), *audsys*(1M), *audusr*(1M), *chfn*(1), *chsh*(1), *passwd*(1), *pwck*(1M), and *init*(1M).

ipccreat Log all IPC create events including *socket*(2), *bind*(2), *ipccreate*(2), and *ipcdest*(2).

ipcopen Log all IPC open events including *connect*(2), *accept*(2), *ipclookup*(2), *ipcconnect*(2), and *ipcrecvcn*(2).

ipcclose Log all IPC close events including *shutdown*(2), and *ipcshutdown*(2).

uevent1 Log user-defined event.

uevent2 Log user-defined event.

uevent3 Log user-defined event.

ipcdgram Log IPC Datagram transactions.

Note that some commands such as *init*(1M) may occur in more than one category because the event varies, depending on the operation done by the command.

AUTHOR

The auditing system described above was developed by HP.

SEE ALSO

audsys(1M), audusr(1M), audevent(1M), audisp(1M), audctl(2), audswitch(2), audwrite(2), getaudid(2), setaudid(2), getevent(2), setevent(2), audit(4).

NAME

context - process context

DESCRIPTION

The context is a set of character strings associated with each process. Each string corresponds to a characteristic of the machine the process is running on. The strings included in the context of every process include:

- cnode name
- types of executable files that can be run by the hardware
- type of cnode ("localroot" or "remoteroot")
- the string "default"

The process context is used to access context dependent files (see *cdf*(4)).

Multiple strings may be present in the context that indicate the ability to run executable code for the designated hardware. When two or more strings appear in the same context, they appear in the order listed.

Series 300 and 400 strings include:

HP-MC68040
HP98248A
HP-MC68881
HP98635A
HP-MC68020
HP-MC68010

Valid Series 700 and 800 strings are:

PA-RISC1.1
HP-PA

Note that presence of a string does not mean that the designated hardware itself is present. For example, since the MC68020 processor supports a superset of the MC68010 instruction set, processes running on a system with an MC68020 processor will have **HP-MC68010** in their context, as well as **HP-MC68020**. The string **HP-MC68010** is present on all series 300 systems.

Similarly, since the PA-RISC version 1.1 processor supports a superset of the HP-PA (PA-RISC version 1.0) instruction set, processes running on a system with a PA-RISC version 1.1 processor will have **HP-PA** in their context, as well as **PA-RISC1.1**.

EXAMPLE

A process running on an HP9000 model 350 workstation, with cnode name **william** could have the following strings in its context:

william
remoteroot
HP-MC68881
HP-MC68020
HP-MC68010
default

Note that this hardware is capable of running executables with the instruction sets for the MC68881, MC68020, and the MC68010. Every process's context ends with the string **default**. Also note that the system call *getcontext*(2) and the command *getcontext*(1) show the context as a single string:

william remoteroot HP-MC68881 HP-MC68020 HP-MC68010 default

WARNINGS

Unless an order is specified, users and applications should not depend on the order of strings within the context. However, **default** is always the last string. Other aspects of this order may vary between releases.

SEE ALSO

getcontext(1), getcontext(2), cdf(4).

AUTHOR

context was developed by HP.

NAME

dirent.h - format of directory streams and directory entries

SYNOPSIS

#include <sys/types.h>
#include <dirent.h>

DESCRIPTION

This header file defines data types used by the directory stream routines described in *directory*(3C).

The following data types are defined:

DIR A structure containing information about an open directory stream.

struct dirent A structure defining the format of entries returned by the *readdir* function (see *directory*(3C)).

The **struct dirent** structure includes the following members:

```
char d_name[MAXNAMLEN+1];   /* name of directory entry */
ino_t d_ino;                /* file serial number */
short d_namlen;             /* length of string in d_name */
short d_reclen;             /* length of this record */
```

The constant **MAXNAMLEN** is defined in **<dirent.h>**.

Note that the **d_reclen** entry is used internally to represent the offset from the current entry to the next valid entry. Therefore, **d_reclen** is not the length of the current entry, but the length of the current record where a record is an entry plus any currently unused space between the current entry and the next valid entry. The unused space between valid **dirent** entries results from changes in a directory's contents, such as the deletion of files and other directories.

This file also contains external declarations for the functions in the *directory*(3C) package.

AUTHOR

dirent.h was developed by AT&T and HP.

SEE ALSO

directory(3C), ndir(5).

STANDARDS CONFORMANCE

`<dirent.h>`: AES, SVID2, XPG2, XPG3, XPG4, FIPS 151-2, POSIX.1

NAME

dld.sl - dynamic loader

DESCRIPTION

The `/lib/dld.sl` program is the dynamic loader. It is invoked automatically at startup time by `/lib/crt0.o` in programs that use shared libraries. The dynamic loader is, itself, a shared library, although it defines no symbols for use by user programs.

Shared Libraries

Shared libraries are executable files created with the `-b` option to `ld` (see *ld*(1)). They contain position-independent code (PIC) that can be mapped anywhere in the address space of a process and executed with a minimum of relocation. PIC can use PC-relative addressing modes and/or linkage tables. It can be generated with the `+z` option to the compilers. See the *Programming on HP-UX* manual for details on writing PIC in assembly language.

Incomplete Executables

When creating an executable (`a.out`) file from object files and libraries, the linker does not copy text from the shared library into the output file. Instead, the dynamic loader maps the library into the address space of the process at run time. The linker binds all program references to shared library routines to entries in a linkage table, and relies on the dynamic loader to fill in the linkage table entries once the libraries have been mapped. This linkage table serves as a jump table. Shared-library data items referenced by the program are copied into the program executable file so that the data references can be resolved statically. The resultant program is called an **incomplete executable**.

Loading

At run time, the dynamic loader attaches to the process all shared libraries that were linked with the program. The text segment of a library is shared among all processes that use it. The data and bss segments are shared on a page-by-page basis. When a process writes to a data or bss page, a modified copy of that page is made for the process.

Binding

The dynamic loader also resolves symbolic references between the executable and libraries. By default, function calls are trapped via the linkage table and bound on first reference. References to variables and other absolute address references cannot be trapped. They are bound on the first resolution of a function call that could potentially reference the object.

If the `-B immediate` option to `ld` is used, the loader binds all necessary references at startup time. This dramatically increases the startup cost of a program, but ensures that no more binding operations will be required later. Thus, better real time response may result, and the risk of a later abort due to unresolved externals is eliminated.

Version Control

Since code from a shared library is mapped at run time from a separate shared library file, modifications to a shared library may alter the behavior of existing executables. In some cases, this may cause programs to operate incorrectly. A means of version control is provided to solve this problem.

Whenever an incompatible change is made to a library interface, both versions of the affected module or modules are included in the library. A mark indicating the date (month/year) the change was made is recorded in the new module via the pragma `HP_SHLIB_VERSION` in C, or the compiler directive `SHLIB_VERSION` in Fortran and Pascal. This date applies to all symbols defined within the module. A high water mark giving the date of the latest incompatible change is recorded in the shared library, and the high water mark for each library linked with the program is recorded in the incomplete executable file.

At run time, the dynamic loader checks the high water mark of each library and loads the library only if it is at least as new as the high water mark recorded at link time. When binding symbolic references, the loader chooses the latest version of a symbol that is not later than the high water mark recorded at link time. These two checks help ensure that the version of each library interface used at run time is the same as was expected at link time.

Explicit Loading And Binding

The duties of the dynamic loader as described above are all performed automatically, although they can be controlled somewhat by appropriate options to `ld`. The dynamic loader can also be accessed programmatically. The reserved variable `__dld_loc`, which is defined in `/lib/crt0.o`, points to a jump table within the dynamic loader. The routines described under *shl_load*(3X) provide a portable interface that

allows the programmer to explicitly attach a shared library to the process at run time, to calculate the addresses of symbols defined within shared libraries, and to detach the library when done.

DIAGNOSTICS

If the dynamic loader is not present, or cannot be invoked by the process for any reason, an error message is printed to standard error and the process terminates with a non-zero exit code.

Normally, the operation of the dynamic loader is visible only when there is a fatal error of some kind. In these cases, an error message is printed to standard error and a `SIGABRT` signal is sent to the process. These errors fall into two basic categories: errors in attaching a shared library, and errors in binding symbols. The former can occur only at process startup time but the latter can occur at any time during process execution unless the `-B immediate` option is used with `ld`. Possible errors that can occur while attaching a shared library include library not present, library not executable, library corrupt, high water mark too low, or insufficient room in the address space for the library. Possible errors that can occur while binding symbols include symbol not found (unresolved external), or library corrupt.

When using the explicit load facilities of the dynamic loader, these types of errors are not considered fatal. Consult *shl_load*(3X) for information on error handling.

WARNINGS

The startup cost of the dynamic loader is significant, even with deferred binding, and can cause severe performance degradation in processes dominated by startup costs (such as simple "hello world" programs). In addition, position-independent code is usually slower than normal code, so performance of a program may be adversely affected by the presence of PIC in shared libraries. However, the advantages of decreased disk space usage and decreased memory requirements for executables should outweigh these concerns in most cases.

There are rare cases where the behavior of a program differs when using shared libraries as opposed to archive libraries. This happens primarily when relying on undocumented and unsupported features of the compilers, assembler, and linker. See the *Programming on HP-UX* manual for more details.

The library developer is entirely responsible for version control and must be thorough in identifying incompatible changes to library interfaces. Otherwise, programs may malfunction unexpectedly with later versions of the library. There is little an application user can do if version control is not handled properly by the library developer. The application developer can usually resolve problems by modifying the source code to use the new interfaces then recompiling and relinking against the new libraries.

DEPENDENCIES

Series 300/400

The Series 300/400 Pascal compiler does not generate PIC, so true shared libraries cannot be created from Pascal source.

Dynamic load libraries that are not completely shared can be created from modules that do not contain position-independent code, as long as the **+l** option to the Pascal compiler is used. The **+s** option to the assembler can also be used to generate modules from non-PIC sources that can be included in dynamic load libraries." The resultant library text segment contains absolute addresses that must be relocated at run time. This causes each text page with code that is not position independent to be modified at run time for each process, which effectively defeats sharing. The text segments of such libraries cannot be made read only.

Series 700/800

Copy-on-write of shared library data and bss pages is not supported; a separate copy of a page is made for each process that references (that is, either writes or reads) it.

AUTHOR

The `/lib/dld.sl` program was developed by HP.

SEE ALSO

as_300(1), as_800(1), cc(1), f77(1), ld(1), pc(1), crt0(3), shl_load(3X), a.out_300(4), a.out_800(4).

Programming on HP-UX manual.

NAME

environ - user environment

DESCRIPTION

An array of strings called the *environment* is made available by *exec*(2) when a process begins. By convention, these strings have the form *name=value*. The following names are used by various commands (listed in alphabetical order):

HOME Name of the user's login directory, set by *login*(1) from the password file (see *passwd*(4)).

LANGOPTS Defines language options for mode and data order in the form:

LANGOPTS=[*mode*][*_order*]

LANGOPTS values are given in English as an ASCII character string. *mode* describes the mode of a file where **l** (ell) represents Latin mode and **n** represents non-Latin mode. Non-Latin mode is assumed for values other than **l** and **n**. *order* describes the data order of a file where **k** is keyboard order and **s** is screen order.

LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, and LC_TIME

Internationalization environment variables corresponding to *setlocale* categories of the same name. They define the user's requirements for language, territory, and codeset with respect to character collation, character classification and conversion, currency symbol and monetary value format, numeric data presentation, and time formats, respectively. If any of these are not defined in the environment, LANG provides the defaults.

Syntax for the environment variables LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, and LC_TIME is:

language[*_territory*][*.codeset*][*@modifier*]

The *@modifier* field allows the user to select between more than one value of a category within the same language definition. For example, to interact with the system in Dutch, but sort German files, the following environment variables must be set in the environment :

LANG=dutch
LC_COLLATE=german

The example could be extended to select "unfolded" collation (see *hpnls*(5)) by use of the *@modifier* field :

LC_COLLATE=german@nofold

At run-time, these values are bound to a program's locale by the *setlocale*() function. See *nlsinfo*(1) for a list of valid modifiers associated with each available language.

The modifier component of the internationalization environment variables can be a maximum of MOD_NAME_SIZE bytes. The remainder of each environment variable can be up to LC_NAME_SIZE bytes in length (see **<locale.h>**).

MANPATH Contains a colon-separated list of directory prefixes to be searched by *man*(1) for manual entries. Upon logging in, **/etc/profile** (or **/etc/csh.login**) sets **MANPATH=/usr/man:/usr/contrib/man:usr/local/man**.

MANPATH uses the same syntax as the PATH environment variable, with the addition of recognizing the specifiers **%L**, **%l**, **%t**, and **%c** as used in the NLSPATH environment variable. See NLSPATH below for a description of these specifiers. This provides a way to specify paths to locale-specific manual entries.

It is assumed that each of the prefixes given in MANPATH contain subdirectories of the form **man*, man*.Z, cat*** and **cat*.Z.** (see *man*(1), *catman*(1M), and *fixman*(1)).

NLSPATH Contains a sequence of pseudo-pathnames used by *catopen*(3C) when attempting to locate message catalogs. Each pseudo-pathname contains a name template consisting of an optional path prefix, one or more substitution field descriptors, a file name and an optional file name suffix. For example, given:

NLSPATH="/system/nlslib/msg.cat"

catopen(3C) attempts to open the file **/system/nlslib/msg.cat** as a message catalog.

Field descriptors consist of a % followed by a single character. Field descriptors and their substitution values are:

%N	The value of the *name* parameter passed to *catopen*(3C).
%L	The value of LANG.
%l	The *language* element from LANG.
%t	The *territory* element from LANG.
%c	The *codeset* element from LANG.
%%	Replaced by a single %.

For example, given:

NLSPATH="/system/nlslib/%L/%N.cat"

catopen(3C) attempts to open the file **/system/nlslib/$LANG/*name*.cat** as a message catalog.

A null string is substituted if the specified value is not defined. Separators are not included in **%t** and **%c** substitutions. Note that a default value is not supplied for **%L**. If LANG is not set and NLSPATH had the value in the previous example, *catopen*(3C) would attempt to open the file **/system/nlslib//*name*.cat** as a message catalog.

Path names defined in NLSPATH are separated by colons (:). A leading colon or two adjacent colons (::) is equivalent to specifying **%N**. For example, given:

NLSPATH=":%N.cat:/nlslib/%L/%N.cat"

catopen(3C) will attempt to open the following files in the indicated order: *./name*, *./name*.**cat**, and **/nlslib/$LANG/***name*.**cat**. The first file successfully opened is taken as the message catalog.

A default pseudo-pathname defined by the system is effectively appended to NLSPATH and used by *catopen*(3C) whenever a message catalog cannot be opened in any of the user defined pseudo-pathnames. This system-wide default path is:

/usr/lib/nls/%l/%t/%c/%N.cat

PAGER PAGER indicates the paginator through which output from certain commands is piped. Its value must be a string specifying the complete command line of the desired paginator. Two examples are:

PAGER="more -cs"

PAGER="pg -c"

PAGER affects several commands, including *man*(1) and the interactive mailers. Some of the affected commands provide alternate means of selecting a pager in case there is a conflict. See the individual manual entries for details.

PATH PATH indicates the sequence of directory prefixes that *sh*(1), *time*(1), *nice*(1), *nohup*(1), and others search when looking for a file known by an incomplete path name. Prefixes are separated by colons (:). *Login*(1) sets **PATH=:/bin:/usr/bin**.

LANG The internationalization environment variable LANG identifies the user's requirements for native language, local customs and coded character set, in the form:

LANG=*language*[*_territory*][*.codeset*]

Values of LANG are given in English as an ASCII character string and should be a supported language name (see *lang*(5)). Native Language Support (NLS) operation is initiated at runtime by calling *setlocale*(3C). The following call to *setlocale* binds the execution of a program to the user's language requirements:

setlocale(LC_ALL,"");

This *setlocale* call initializes the program locale from the environment variables associated with *setlocale*. LANG provides the necessary defaults if any of the category-specific environment variables are not set or set to the empty string. In addition, data exists which belongs only to the LC_ALL category; it will always be initialized by LANG.

The LANG environment variable is also used to locate message catalogues. See NLSPATH below.

The LANG environment variable can have a maximum length of SL_NAME_SIZE bytes (see header file **<locale.h>**).

TERM TERM identifies the kind of terminal for which output is to be prepared. This information is used by commands such as *vi*(1) and *mm*(1), which can exploit special capabilities of that terminal.

TZ TZ sets time zone information. TZ can be set using the format:

[:]**STD***offset*[**DST**[*offset*][,*rule*]]

where:

STD and **DST** Three or more bytes that designate the standard time zone (**STD**) and summer (or daylight-savings) time zone (**DST**) **STD** is required. If **DST** is not specified, summer time does not apply in this locale. Any characters other than digits, comma (,), minus (-), plus (+), or ASCII NUL are allowed.

offset *offset* is the value that must be added to local time to arrive at Coordinated Universal Time (UTC). *Offset* is of the form :

hh[:*mm*[:*ss*]]

Hour (*hh*) is any value from 0 through 23. The optional minutes (*mm*) and seconds (*ss*) fields are a value from 0 through 59. The hour field is required. If *offset* is preceded by a -, the time zone is east of the Prime Meridian. A + preceding *offset* indicates that the time zone is west of the Prime Meridian. The default case is west of the Prime Meridian.

rule *rule* indicates when to change to and from summer (daylight-savings) time. The *rule* has the form :

date/time,*date/time*

where the first *date/time* specifies when to change from standard to summer time, and the second *date/time* specifies when to change back. The *time* field is expressed in current local time.

The form of *date* should be one of the following :

J*n* Julian day *n* (1 through 365). Leap days are not counted. February 29 cannot be referenced.

n The zero-based Julian day (0 through 365). Leap days are counted. February 29 can be referenced.

Mm.n.d The *d* day (0 through 6) of week *n* (1 through 5) of month *m* (1 through 12) of the year. Week 5 refers to the last day *d* of month *m*. Week 1 is the week in which the first day of the month falls. Day 0 is Sunday.

time *Time* has the same format as *offset* except that no leading sign ("-" or "+") is allowed. The default, if *time* is not given, is 02:00:00.

While the STD field and the offset field for STD must be specified, if the DST field is also provided, the system will supply default values for other fields not specified. These default values come from file **/usr/lib/tztab** (see *tztab(4)*), and, in general, reflect the various historical dates for start and end of summer time.

Additional names may be placed in the environment by the *export* command and "name=value" arguments in *sh*(1), or by *exec*(2). It is unwise to add names that conflict with the following shell variables frequently exported by **.profile** files: **MAIL**, **PS1**, **PS2** and **IFS**.

The environment of a process is accessible from C by using the global variable:

char **environ;

which points to an array of pointers to the strings that comprise the environment. The array is terminated by a null pointer.

WARNINGS

Some HP-UX commands and library routines do not use the LANG, LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, LC_TIME, or LANGOPTS environment variables. Some commands do not use message catalogs, so NLSPATH does not affect their behavior. See the EXTERNAL INFLUENCES section of specific commands and library routines for implementation details.

NOTES

Coordinated Universal Time (UTC) is equivalent to Greenwich Mean Time (GMT).

AUTHOR

Environ was developed by AT&T and HP.

SEE ALSO

env(1), login(1), sh(1), exec(2), catopen(3C), ctime(3C), getenv(3C), nl_init(3C), profile(4), lang(5), term(5), tztab(4).

STANDARDS CONFORMANCE

`environ`: AES, XPG2, XPG3, XPG4, FIPS 151-2, POSIX.1

NAME

fcntl - file control options

SYNOPSIS

#include <sys/types.h>
#include <fcntl.h>

DESCRIPTION

The *fcntl*(2) function provides for control over open files. This include file describes *requests* and *arguments* to *fcntl* and *open*(2).

Access modes set by *open*(2) and accessed by *fcntl*(2):

O_RDONLY
O_WRONLY
O_RDWR

Mask for file access modes:

O_ACCMODE

File status flags set by *open*(2) or *fcntl*(2) and accessed by *fcntl*(2):

O_NDELAY	Non-blocking I/O
O_NONBLOCK	POSIX-style non-blocking I/O
O_APPEND	Append (writes guaranteed at the end)
O_SYNCIO	Do write through caching

Flag values accessible only to *open*(2):

O_CREAT	Open with file create (uses third open arg)
O_TRUNC	Open with truncation
O_EXCL	Exclusive open
O_NOCTTY	Do not assign a controlling terminal

Requests for *fcntl*(2):

F_DUPFD	Duplicate fildes
F_GETFD	Get file descriptor flags
F_SETFD	Set file descriptor flags
F_GETFL	Get file flags
F_SETFL	Set file flags
F_GETLK	Get blocking file lock
F_SETLK	Set or clear file locks and fail on busy
F_SETLKW	Set or clear file locks and wait on busy

File descriptor flags for F_GETFD, F_SETFD:

FD_CLOEXEC

File segment locking control structure, **struct flock**, including the following members:

```
short   l_type;     /* F_RDLCK, F_WRLCK or F_UNLCK */
short   l_whence;   /* Flag - see lseek(2) */
off_t   l_start;    /* Relative offset in bytes */
off_t   l_len;      /* Size; if 0 then until EOF */
pid_t   l_pid;      /* By F_GETLK - process holding lock */
```

File segment locking types:

F_RDLCK	Read lock
F_WRLCK	Write lock
F_UNLCK	Remove locks

SEE ALSO
fcntl(2), open(2).

STANDARDS CONFORMANCE
<fcntl.h>: AES, XPG2, XPG3, XPG4, FIPS 151-2, POSIX.1

NAME

hier - file system hierarchy

DESCRIPTION

The following outline gives a quick tour through a representative HP-UX directory hierarchy. Some of the directories listed only appear with HP-UX versions that support certain optional commands or packages that use those directories. Some HP-UX versions add special directories not shown here.

/	Root directory.
/bin	Frequently-used commands and those required to boot, restore, recover, and/or repair the system.
/dev	Special files (device files); see *mknod*(1).
/etc	
/etc/newconfig	New (updated) versions of customizable (localizable) configuration files and shell scripts. Shipped here so as not to overwrite current versions. Copied to regular locations for newly installed systems. System administrators may wish to keep them for later reference.
/lib	Frequently-used object code libraries and related utilities.
/lost+found	For connecting detached files; for use by *fsck*(1).
/rbin	An analog to **/bin** for users in the restricted environment of *rsh*(1).
/tmp	Place to put temporary files (those normally with short lifetimes and which may be removed without notice).
/users	User home directories; sometimes immediate, sometimes at lower levels.
/users/guest	Default home directory for user "guest"; see *passwd*(4). Directory exists for novice users; you may wish to remove it.
/usr	Less-frequently-used commands and other miscellaneous things; historically, often a separate, mounted volume.
/usr/adm	
/usr/adm/sa	
/usr/bin	Less-frequently-used commands and those not required to boot, restore, recover, and/or repair the system.
/usr/bin/graph	*Gutil*(1) graphics commands.
/usr/contrib	User-contributed (unsupported, internal) commands, files, etc. Files under this directory come from outside the local site or organization, e.g. from users groups, HP service engineers, etc. See **/usr/local** for local-site commands and files.
/usr/contrib/bin	User-contributed commands.
/usr/contrib/games	User-contributed games.
/usr/contrib/include	User-contributed include files. To include them, you must (in C) give a complete pathname, for example, #include "/usr/contrib/include/symtab.h".
/usr/contrib/lib	User-contributed libraries.
/usr/contrib/man/cat[1-8]	User-contributed manual entries, formatted.
/usr/contrib/man/man[1-8]	User-contributed manual entries, unformatted.
/usr/contrib/man/$LANG/cat[1-8]	User-contributed manual entries, formatted form for installed native languages. The LANG environment variable may take on values given in the **/usr/lib/nls/config** table.

/usr/contrib/man/$LANG/man[1-8]
: User-contributed manual entries, unformatted form for installed native languages.

/usr/include
: High-level C-language header files (shared definitions).

/usr/include/sys
: Low-level (kernel-related) C-language header files.

/usr/lib
: Less-frequently-used object code libraries, related utilities, miscellaneous data files, etc.

/usr/lib/acct
: Certain system-administrative commands.

/usr/lib/cron
: For *cron*(1M) and *at*(1) scheduling information.

/usr/lib/graphics/c
: Device-independent Graphics Library (DGL) special C-language include files. Optional on some systems.

/usr/lib/graphics/demos
: DGL demonstration software.

/usr/lib/graphics/fortran
: DGL special FORTRAN-language include files.

/usr/lib/graphics/pascal
: DGL special Pascal-language include files.

/usr/lib/help
: Data files for *help*(1).

/usr/lib/lex
: Data files for *lex*(1).

/usr/lib/macros
: Macro definition packages for *nroff*(1) and *troff*.

/usr/lib/nlio
: Native Language I/O.

/usr/lib/nls
: Native Language support.

/usr/lib/nls/config
: Correspondence between integer language id and name.

/usr/lib/nls/$LANG
: Language definition (Character Set Support, Local Customs, and Messages) for installed native languages. The LANG environment variable may take on values given in the **/usr/lib/nls/config** table.

/usr/lib/sa

/usr/lib/spell
: Data files for *spell*(1).

/usr/lib/tabset
: Data files to set tabstops.

/usr/lib/term
: Terminal initialization files.

/usr/lib/tmac
: Macro definition packages for *nroff*(1) and *troff*.

/usr/lib/uucp[/*]
: Commands, configuration files, and working directories for *uucp*(1).

/usr/local
: Site-local commands, files, etc. Files under this directory come from inside the local site or organization. See **/usr/contrib** for non-local unsupported commands and files.

/usr/local/bin
: Site-local commands.

/usr/local/games
: Site-local games.

/usr/local/include
: Site-local include files. To include them, you must (in C) give a complete pathname, for example, #include "/usr/local/include/symtab.h".

/usr/local/lib
: Site-local libraries.

/usr/local/man/cat[1-8]
: Site-local manual entries, formatted.

/usr/local/man/man[1-8]
: Site-local manual entries, unformatted.

/usr/local/man/$LANG/cat[1-8]
: Site-local manual entries, formatted form for installed native languages. The LANG

environment variable may take on values given in the **/usr/lib/nls/config** table.

/usr/local/man/$LANG/man[1-8]
: Site-local manual entries, unformatted form for installed native languages.

/usr/mail
: User mailboxes.

/usr/man
: Online documentation.

/usr/man/cat[1-8]
: Optional formatted (by nroff) versions of online documentation for use by *man*(1).

/usr/man/man[1-8]
: Unformatted (nroff-compatible source) versions of online documentation for use by *man*(1).

/usr/man/$LANG
: Online documentation for installed native languages. The LANG environment variable may take on values given in the **/usr/lib/nls/config** table.

/usr/man/$LANG/cat[1-8]
: Formatted native language versions of online documentation for use by *man*(1).

/usr/man/$LANG/man[1-8]
: Unformatted native language versions of online documentation for use by *man*(1).

/usr/news
: Local-system news articles for *news*(1).

/usr/preserve
: Place where *ex*(1) and *vi*(1) save lost edit sessions until recovered.

/usr/rbin
: An analog to **/usr/bin** for users in a restricted environment (as imposed by *rsh*(1)).

/usr/spool
: Spooled (queued) files for various programs.

/usr/spool/cron
: Spooled jobs for *cron*(1M) and *at*(1).

/usr/spool/cron/atjobs
: Spooled jobs for *at*(1).

/usr/spool/lp
: Control and working files for *lp*(1).

/usr/spool/lp/class
: Printer class definition files.

/usr/spool/lp/interface
: Printer interface shell scripts.

/usr/spool/lp/member
: Printer class member definition files.

/usr/spool/lp/request
: Spool directories for each logical destination.

/usr/spool/uucp
: Queued work, lockfiles, logfiles, status files, and other files for *uucp*(1).

/usr/spool/uucppublic[/*]
: Publicly-accessible directory for use with *uucp*(1).

/usr/src
: Source files. Only present on HP-UX implementations which support source.

/usr/src/cmd/*
: Source for commands. Simple command sources reside at the top level. Subdirectories are named after specific commands, e.g. **/usr/src/cmd/cc**, and contain the source for multi-file or otherwise complicated commands. Directory structure below here depends on the individual command; see the associated makefiles.

/usr/src/games/*
: Source for games. Simple game sources reside at the top level. Subdirectories are named after specific games, e.g. **/usr/src/games/master**, and contain the source for multi-file or otherwise complicated games. Directory structure below here depends on the individual game; see the associated makefiles.

/usr/src/head
: Include files which are copied into **/usr/include/***.

/usr/src/lib
: Source for libraries, in many subdirectories.

/usr/src/lib/libF77
: Source for FORTRAN-77 miscellaneous (mostly math) libraries.

/usr/src/lib/libI77
: Source for FORTRAN-77 I/O libraries.

/usr/src/lib/libPW	Source for Programmer's Workbench libraries.
/usr/src/lib/libc	Source for standard C libraries.
/usr/src/lib/libcurses/*	Source for curses (cursor control) libraries.
/usr/src/lib/libl	Source for *lex*(1) libraries.
/usr/src/lib/libm	Source for C math libraries.
/usr/src/lib/liby	Source for *yacc*(1) libraries.
/usr/tmp	Alternate place to put temporary files; usually used when there may be very many of them or if they will be large.

DEPENDENCIES

Some directories include commands or files not supported on all HP-UX implementations.

SEE ALSO

find(1), grep(1), ls(1), whereis(1).

NAME

hostname - host name resolution description

DESCRIPTION

Hostnames are domains. A domain is a hierarchical, dot-separated list of subdomains. For example, the machine **monet,** in the **Berkeley** subdomain of the **EDU** subdomain of the ARPANET would be represented as

```
monet.Berkeley.EDU
```

(with no trailing dot).

Hostnames are often used with network client and server programs, which must generally translate the name to an address for use (this function is typically performed by the library routine *gethostbyname*(3N)). Hostnames are resolved by the internet name resolver in the following fashion. When using NIS or the host table, the hostname is looked up without modification.

If the name consists of a single component, i.e. contains no dot, and if the environment variable **HOSTALIASES** is set to the name of a file, that file is searched for a string matching the input hostname. The file should consist of lines made up of two strings separated by white-space, the first of which is the hostname alias, and the second of which is the complete hostname to be substituted for that alias. If a match is found between the hostname to be resolved and the first field of a line in the file (uppercase and lowercase are treated as equivalent for 7-bit ASCII), the substituted name is looked up with no further processing.

If the input name ends with a trailing dot, the trailing dot is removed and the remaining name is looked up with no further processing.

If the input name does not end with a trailing dot, it is looked up by searching through a list of domains until a match is found. If the name has not been matched by searching through a list of domains and the name contains a dot, it is looked up without modification. The default search list includes first the local domain, then its parent domains with at least two name components (longest first). For example, in the domain **CS.Berkeley.EDU,** the name **lithium.CChem** will be checked first as **lithium.CChem.CS.Berkeley.EDU** and then as **lithium.CChem.Berkeley.EDU. Lithium.CChem.EDU** will not be tried, as there is only one component remaining from the local domain. If none of the previous attempts matched, the name would be looked up as **Lithium.CChem** without any domains appended.

AUTHOR

hostname was developed by the University of California, Berkeley.

SEE ALSO

named(1M), gethostbyname(3N), resolver(3N),

RFC1034, RFC1035

NAME

hpnls - HP Native Language Support (NLS) Model

DESCRIPTION

Native Language Support (NLS) reduces or eliminates the barriers that would otherwise make HP-UX difficult to use in a non-English-speaking work environment. NLS is available at the user-command level as well as through commands and libraries that can be used to develop international software applications.

Many existing C library routines have been modified to operate based upon a program's locale. A locale is the run-time NLS environment of a program which is loaded by `setlocale()` (see *setlocale*(3C)). For a complete list of what library routines are affected by `setlocale()`, see *setlocale*(3C)).

In addition to routines that operate based on the program's locale, there are also commands and routines to provide a messaging system for accessing program messages based on the language requirements of the end-user.

Many HP-UX commands have been modified to operate in a manner sensitive to the language requirements of the end-user. These language requirements are established through the internationalization environment variables (see *environ*(5)). The EXTERNAL INFLUENCES/Environment Variables section of the manual entry for each command that has NLS capabilities describes which environment variables the command is sensitive to.

In addition, the `portnls` routines are a set of library routines that perform miscellaneous language-dependent operations. `portnls` is intended to provide portability between HP-UX and MPE (another HP operating system). See *portnls*(5) for more information.

Below are areas of functionality that are considered language-sensitive :

Character Handling

NLS provides for handling characters outside the 7-bit USASCII codeset. Most languages require a minimum of 8-bits to support all the characters needed to communicate in that language. Characters must be handled according to the requirements of the language they represent.

Codesets with 8-bit characters have been defined to support phonetic languages, such as the Western European languages. The use of an 8-bit character allows for an additional 128 characters beyond the USASCII codeset.

More than 8 bits are needed to uniquely define codes for characters required by ideographic languages such as Japanese. For such languages, multibyte codesets are used in which a character is represented by a sequence of one or more bytes. Multibyte codesets are defined according to the rules of a multibyte encoding scheme. Encoding schemes define the particular sequences of byte values that can be used to form characters. The EUC encoding scheme is supported by HP-UX. However, only the one- and two-byte forms of EUC are currently supported. Refer to the *Native Language Support* User's Guide for more information about EUC.

Character Classification

Characters have many attributes associated with them. For example, characters may be classified as printable, alphabetic, numeric, etc. These attributes are commonly referred to as ctype characteristics. Characters and their associated attributes differ between languages. Character processing that depends on character classification must be sensitive to these differences.

Shifting

The notion of uppercase and lowercase differs between languages. For example, in some languages accents are discarded when characters are shifted to uppercase. Some languages have no notion of uppercase and lowercase characters. For example, shifting a character has no effect in ideographic languages.

Collating

Collating sequences differ between languages and most languages require multiple collating sequences. The following collation features are available to provide a full "dictionary-" or "context-based" language-dependent comparison :

Two-to-one conversions

Some languages, such as Spanish, require two adjacent characters to occupy one position in the collating sequence. Examples are **CH** (which follows **C**) and **LL**

(which follows **L**).

One-to-two conversions
Some languages, such as German, require one character (such as "sharp S") to occupy two adjacent positions in the collating sequence.

Don't-care characters
Some languages designate certain characters to be ignored in character comparisons. For example, if - is a don't-care character, the strings **REACT** and **RE-ACT** would equal each other when compared.

Uppercase/lowercase and accent priority
Many languages require a "two-pass" collating algorithm. In the first pass, accents are stripped from their letters and the resulting two strings are compared. If they are equal, a second pass with the accents reinserted is performed to break the tie. Uppercase/lowercase differences can also be first ignored then used to break ties in this fashion.

Two common methods of collation for phonetic languages are *folded* and *nonfolded*. A folded collating sequence is made up of the uppercase and lowercase characters intermixed. An unfolded collating sequence is made up of all the uppercase characters followed by the lowercase characters. For example, collating the characters **a b c A B C** with folded collation would result in the following order :

```
A a B b C c
```

Collating the same characters with unfolded collation would result in the following order :

```
A B C a b c
```

For languages in which folded and unfolded collation methods are defined, HP-UX uses folded as the default. The **setlocale** modifier **nofold** can be used to enable the nonfolded collating method (see *environ*(5)). The **nlsinfo** command reports the collating methods supported for each language (see *nlsinfo*(1)).

Directionality
Two properties of text files and Native Languages must be understood to process text in non-Western languages. They are the mode of the language, and the order of the characters.

Mode refers to the direction that a language is naturally read. European languages read from left to right, some Middle Eastern languages read from right to left, and Far Eastern languages usually use vertical columns, beginning from the right.

Order describes the order in which characters are written, stored in a file, or displayed. Keyboard order refers to the order of keystrokes by a user. Screen order refers to the order in which characters are displayed on a terminal screen or printed.

Screen order can differ from keyboard order when using a terminal that supports mixing Latin and non-Latin text, each requiring different directionality. In the following example, the text mode is right-to-left; *n* represents a non-Latin character, *l* represents a Latin character, and the numbers represent the order in which the sequence is typed.

In keyboard order, the letters would be stored in a file as follows:

n1 n2 n3 l4 l5 l6

In screen order, the letters would be stored in a file as follows:

n1 n2 n3 l6 l5 l4

However, both screen-order and key-order sequences would look identical on the screen because the terminal would be configured to display the characters properly according to the directionality requirements of both the Latin and non-Latin languages.

Local Customs
NLS supports customs that are specific to a particular geographic region such as representation of numeric and monetary data, date, and time. These customs can differ not only between languages, but also between regions that share a common language.

Representation of numbers
The character used to denote the radix of a decimal number varies for different

regions. Similarly the use of a "thousands" indicator or grouping of digits can vary with local custom. Characters used to represent digits can also vary for different regions.

Monetary representation
The currency symbol and the formatting of monetary values varies from country to country. For instance, the symbol can either precede or follow the monetary value. Some currencies allow decimal fractions while others use alternate methods of representing smaller monetary values.

Date and time representation
While the Gregorian calendar is most common, some countries use other methods for determining meridian day and year, usually based on seasonal, astronomical, or historical events. Month and weekday names as well as the format of date and time varies from country to country. Even when a strictly numeric date/time representation is used, the order of year, month and day, and the delimiters that separate them, is not universal.

The HP-UX system clock runs on Coordinated Universal Time. Time zone adjustments for a particular regions can be specified through the `TZ` environment variable (see *environ*(5)).

Messages
Messages issued by a program should be sensitive to the language of the end-user. NLS provides a messaging facility for extracting hard-coded strings (messages) from an application source code and storing them externally to the code. Utilities are provided to aid the translation of messages such that at runtime the program accesses messages that coincide with the end-user's native language.

FILES
`/usr/lib/nls/*`

AUTHOR
`hpnls` was developed by HP.

SEE ALSO
insertmsg(1), gencat(1), catgets(3C), catopen(3C), setlocale(3C), wconv(3X), wctype(3X), wstring(3X), environ(5), lang(5).

Native Language Support User's Guide.

For additional information, see the EXTERNAL INFLUENCES/Environment Variables section of applicable manual entries for commands and library routines.

NAME

ioctl - generic device control commands

SYNOPSIS

#include <sys/ioctl.h>
ioctl(fildes, request, arg)
int fildes, request;

DESCRIPTION

The *ioctl*(2) system call provides for control over open devices. This include file describes *requests* and *arguments* used in *ioctl*(2) which are of a generic nature. For details about how individual requests will affect any particular device, see the corresponding device manual entry in Section (7). If a device does not support an ioctl request it returns **EINVAL**.

FIONREAD

Returns in the long integer whose address is *arg* the number of characters immediately readable from the device file.

FIOSSAIOSTAT

For those character device files which support this command, if the integer whose address is *arg* is non-zero, system asynchronous I/O is enabled; that is, enable SIGIO to be sent to the process currently designated with FIOSSAIOOWN (see below) whenever device-file-dependent events occur. If no process has been designated with FIOSSAIOOWN, then enable SIGIO to be sent to the first process to open the device file.

If the designated process has exited, the SIGIO signal is not sent to any process.

If the integer whose address is *arg* is 0, system asynchronous I/O is disabled.

FIOGSAIOSTAT

For those character device files which support this command, the integer whose address is *arg* is set to 1, if system asynchronous I/O is enabled. Otherwise, the integer whose address is *arg* is set to 0.

FIOSSAIOOWN

For those character device files which support this command, set process ID to receive the SIGIO signals with system asynchronous I/O to the value of the integer whose address is *arg*. Users with appropriate privileges can designate that any process receive the SIGIO signals. If the request is not made by the super-user, only the calling process is allowed to designate that itself or another process whose real or saved effective user ID matches its real or effective user ID, or a process which is a descendant of the calling process, receive the SIGIO signals. If no process can be found corresponding to that specified by the integer whose address is arg, the call will fail, with **errno** set to ESRCH. If the request is not made by the super-user and the calling process attempts to designate a process other than itself or (1) another process whose real or saved effective user ID matches its real or effective user ID, or (2) a process which is not a descendant of the calling process, the call fails, with **errno** set to EPERM.

If the designated process subsequently exits, the SIGIO signal will not be sent to any process.

The default when opening a device file is that the process performing the open is set to receive the SIGIO signals.

FIOGSAIOOWN

For those character device files which support this command, the integer whose address is *arg* is set to the process ID designated to receive SIGIO signals.

FIOSNBIO

For those character device files which support this command, if the integer whose address is *arg* is non-zero, non-blocking I/O is enabled; that is, subsequent reads and writes to the device file are handled in a non-blocking manner (see below). If the integer whose address is *arg* is 0, non-blocking I/O is disabled.

For reads, non-blocking I/O prevents all read requests to that device from blocking, whether the requests succeed or fail. Such read requests complete in one of three ways:

- If there is enough data available to satisfy the entire request, the read completes successfully, having read all of the data, and returns the number of bytes read;
- If there is not enough data available to satisfy the entire request, the read completes successfully, having read as much data as possible, and returns the number of bytes it was able to read;
- If there is no data available, the read fails and **errno** is set to EWOULDBLOCK.

For writes, non-blocking I/O prevents all write requests to that device file from blocking, whether the requests succeed or fail. Such a write request completes in one of three ways:

- If there is enough space available in the system to buffer all the data, the write completes successfully, having written out all of the data, and returns the number of bytes written;
- If there is not enough space in the buffer to write out the entire request, the write completes successfully, having written as much data as possible, and returns the number of bytes it was able to write;
- If there is no space in the buffer, the write fails and **errno** is set to EWOULDBLOCK.

To prohibit non-blocking I/O from interfering with the O_NDELAY flag (see *open*(2) and *fcntl*(2)), the functionality of O_NDELAY always supercedes the functionality of non-blocking I/O. This means that if O_NDELAY is set, the driver performs read requests in accordance with the definition of O_NDELAY. When O_NDELAY is not set, the definition of non-blocking I/O applies.

The default on open of a device file is that non-blocking I/O is disabled.

FIOGNBIO
: For those character device files which support this command, the integer whose address is *arg* is set to 1, if non-blocking I/O is enabled. Otherwise, the integer whose address is *arg* is set to 0.

WARNINGS

FIOSSAIOSTAT is similar to 4.2 BSD FIOASYNC, with the addition of provisions for security. FIOGSAIOSTAT is of HP origin, complements FIOSSAIOSTAT, and allows saving and restoring system asynchronous I/O TTY state for BSD style job control. FIOSSAIOOWN is similar to 4.2 BSD FIOSETOWN, with the addition of provisions for security. FIOGSAIOOWN is similar to 4.2 BSD FIOGETOWN. Note also the difference that the 4.2 BSD version of this functionality used process groups, while the HP-UX version only uses processes. FIOSNBIO is the same as 4.2 BSD FIONBIO, except that it does not interfere with the AT&T O_NDELAY *open* and *fcntl* flag. FIOGNBIO is of HP origin, complements FIOSNBIO, and allows saving and restoring non-blocking I/O TTY state for BSD-style job control.

SEE ALSO

ioctl(2).
Section 7 of this manual.

NAME

lang - description of supported languages

DESCRIPTION

HP-UX NLS (Native Language Support) provides support for the processing and customs requirements of a variety of languages. To enable NLS support for a particular language, a language definition must exist on the HP-UX system. The `nlsinfo` command (see *nlsinfo*(1)) displays information regarding what languages are currently supported on a particular HP-UX system. In addition, `nlsinfo` also provides information on what modifiers (see *environ*(5)) are valid for each available language.

The default processing language for HP-UX is `POSIX`. `POSIX` provides an environment in which processing occurs without NLS functionality. This environment is based on the 7-bit-coded USASCII character set.

WARNINGS

Previous releases of HP-UX provided `n-computer` as the default processing language. For backward-compatibility, `n-computer` remains the default processing language when `nl_init()` is used (see *nl_init*(3C)). `POSIX` and `C` are equivalent and can be used interchangeably. `POSIX/C` and `n-computer` are equivalent with the exception of the *nl_langinfo*(3C) items currency symbol (`CRNCYSTR`) and thousands seperator (`THOUSEP`). `POSIX/C` defines both to be the empty string, while `n-computer` defines the currency symbol to be `$` and the thousands separator to be `,`. `nl_init()` and `n-computer` are provided for backward-compatibility. Use `setlocale()` instead (see *setlocale*(3C)).

The NLS environment can also be initialized by passing a language ID number to routines that accept a *langid* parameter. The language ID number corresponds to a language name. The language ID numbers and the corresponding language names are stored in file `/usr/lib/nls/config`. The language ID number and the routines accepting the *langid* parameter are provided for historical reasons only. Routines that provide equivalent functionality without the *langid* parameter are recommended. The WARNINGS section of manual entries for applicable routines indicate what routine to use.

AUTHOR

`lang` was developed by HP.

SEE ALSO

nlsinfo(1), setlocale(3C), wctype(3X), environ(5), hpnls(5)

NAME

langinfo - language information constants

SYNOPSIS

```
#include <langinfo.h>
```

DESCRIPTION

This header file contains the constants used to identify items of langinfo data (see *nl_langinfo*(3C)). The mode of *items* is given in `<nl_types.h>`. The following constants are defined (CATEGORY indicates in which *setlocale*(3C) category each item is defined):

Constant	Category	Description
`D_T_FMT`	`LC_TIME`	String for formatting the `%c` (date and time) directive of *date*(1), *getdate*(3C), and *strftime*(3C).
`D_FMT`	`LC_TIME`	String for formatting the `%x` (date) directive of *date*(1), *getdate*(3C), and *strftime*(3C).
`T_FMT`	`LC_TIME`	String for formatting the `%X` (time) directive of *date*(1), *getdate*(3C), and *strftime*(3C).
`DAY_1`	`LC_TIME`	Name of the first day of the week ("Sunday" in English).
:	:	:
`DAY_7`	`LC_TIME`	Name of the seventh day of the week.
`ABDAY_1`	`LC_TIME`	Abbreviated name of the first day of the week ("Sun" in English).
`ABDAY_7`	`LC_TIME`	Abbreviated name of the seventh day of the week.
`MON_1`	`LC_TIME`	Name of the first month in the Gregorian year.
:	:	:
`MON_12`	`LC_TIME`	Name of the twelfth month.
`ABMON_1`	`LC_TIME`	Abbreviated name of the first month.
:	:	:
`ABMON_12`	`LC_TIME`	Abbreviated name of the twelfth month.
`RADIXCHAR`	`LC_NUMERIC`	Radix character ("decimal point" in English). The string returned is the same as the `decimal_point` element in the structure returned by *localeconv*(3C).
`THOUSEP`	`LC_NUMERIC`	Separator for thousands. The string returned is the same as the `thousands_sep` element in the structure returned by *localeconv*(3C).
`YESEXPR`	`LC_MESSAGES`	Affirmative response for expression.
`NOEXPR`	`LC_MESSAGES`	Negative response for expression.
`YESSTR`	`LC_MESSAGES`	Affirmative response for yes/no questions. (Obsolete: use `YESEXPR`)
`NOSTR`	`LC_MESSAGES`	Negative response for yes/no questions. (Obsolete: use `NOEXPR`)
`CRNCYSTR`	`LC_MONETARY`	Symbol for currency preceded by "–" if it precedes the number, "+" if it follows the number, and "." if it replaces the radix. For example, "–DM" would be used for German (DM1234,56), "+ Kr" for Danish (1234,56 Kr), and ".$" for Portuguese (1234$56). See *localeconv*(3C) for alternative currency formatting information.
`BYTES_CHAR`	`LC_CTYPE`	Maximum number of bytes per character for the character set used for the specified language. For example, "1" for English and most European languages, and "2" for Japanese and several other Asian languages. This constant is an HP proprietary item and may not be portable to other platforms.

DIRECTION	**LC_ALL**	Value to indicate text direction. Values currently defined include "null", "0" and "1". Values of "null" or "0" indicate that characters are arranged from left-to-right within a line and lines are arranged from top-to-bottom. A value of "1" indicates that characters are arranged from right-to-left within a line and lines are arranged from top-to-bottom. This constant is an HP proprietary item and may not be portable to other platforms.
ALT_DIGIT	**LC_NUMERIC**	A string of the characters that are mapped into the ASCII equivalent string "0123456789b+-.,eE" (where b is a blank). This is also the reverse mapping for output. It is not assumed that the character code values of digits are contiguous or that they are one byte values. A null value for the string indicates that the language has no alternative digits. This constant is an HP proprietary item and may not be portable to other platforms.
ALT_PUNCT	**LC_CTYPE**	A string of the characters that are mapped into the ASCII equivalent string "b!"#$%&'()*+,-./:;<=>?@[\]^_'{|}~" (where b is a blank) in American usage. This is also the reverse mapping for output. It is not assumed that the character code values of punctuation characters are contiguous or that they are one byte values. If any punctuation characters do not have equivalent alternatives, ASCII codes are used in the alternative punctuation string. A null value for the string indicates that the language has no alternative punctuation characters. This constant is an HP proprietary item and may not be portable to other platforms.
AM_STR	**LC_TIME**	Ante meridiem string used with 12-hour time formats ("AM" in English)
PM_STR	**LC_TIME**	Post meridiem string used with 12-hour time formats ("PM" in English)
YEAR_UNIT	**LC_TIME**	Symbol for year. This is usually required to specify date for Asian languages.
MON_UNIT	**LC_TIME**	Symbol for month.
DAY_UNIT	**LC_TIME**	Symbol for day.
HOUR_UNIT	**LC_TIME**	Symbol for hour. This is usually required to specify time for Asian languages.
MIN_UNIT	**LC_TIME**	Symbol for minute.
SEC_UNIT	**LC_TIME**	Symbol for second.
ERA_D_FMT	**LC_TIME**	Default string for formatting the **%E** (Emperor/Era name and year) directive of *date*(1) and *strftime*(3C) if an individual era format is not specified for an era (see *localedef*(1M)).
T_FMT_AMPM	**LC_TIME**	Time representation in the 12-hour clock format with **AM_STR** and **PM_STR**.

WARNINGS

It is recommended to use **strftime()** to access date and time information defined in **category** (see *strftime*(3C)), **LC_TIME** and *localeconv*(3C) to access information corresponding to **RADIXCHAR**, **THOUSEP**, and **CRNCYSTR** (see *localeconv*(3C)).

AUTHOR

langinfo was developed by HP.

SEE ALSO

date(1), getdate(3C), langinfo(3C), localeconv(3C), nl_langinfo(3C), setlocale(3C), strftime(3C), hpnls(5), lang(5).

NAME

limits - implementation-specific constants

SYNOPSIS

#include <limits.h>

DESCRIPTION

The following symbols are defined in **<limits.h>** and are used throughout the descriptive text of this manual. The column headed **HP-UX Value** lists the values that application writers should assume for portability across all HP-UX systems.

Symbols after values are interpreted as follows:

- **+** Actual limit might be greater than specified value on certain HP-UX systems.
- **-** Actual limit might be less than the specified value on certain HP-UX systems.
- **=** Actual limit is always equal to the specified value and does not vary across HP-UX systems.
- ***** The name of this limit is defined *only* if the preprocessor macro _XPG2 is defined, either by the compilation flag -D_XPG2, or by a **#define** directive in the source before **<limits.h>** is included in the source.
- **#** The value defined for this limit might not be a compile-time constant. The value defined always evaluates to an integer expression at run time.

Some of these limits vary with system configuration, and can be determined dynamically by using *sysconf*(2). Others can vary according to file system or device associated with a specific file, and can be determined with *pathconf*(2). Others are obsolescent because they are redundant with other limits or not useful in portable applications. They are provided only for importability of applications from other systems, to support applications that comply with the *X/Open Portability Guide, Issue 2*, and for backward compatibility with earlier versions of HP-UX. The _XPG2 flag should not be defined in new applications.

By including the **<limits.h>** file in the compilation an application can test the appropriate limits to determine whether it can operate on a particular system, or it might even alter its behavior to match the system to increase its portability across a varying range of limit settings and systems.

Constant	Description	HP-UX Value
ARG_MAX	Max length of arguments to *exec*(2) in bytes, including environment data	5120 +*
CHAR_BIT	Number of bits in a **char**	8 =
CHAR_MAX	Max integer value of a **char**	127 =
CHAR_MIN	Min integer value of a **char**	-128 =
CHILD_MAX	Max number of simultaneous processes per user ID	25 +-*
CLK_TCK	Number of clock ticks per second	50 +#
DBL_DIG	Digits of precision of a **double**	16 +
DBL_MAX	Max positive value of a **double**	1.7976931348623157e+308+
DBL_MIN	Min positive value of a **double**	4.94065645841246544e-324 -
FCHR_MAX	Max file offset in bytes	INT_MAX +-*
FLT_DIG	Digits of precision of a **float**	6 +
FLT_MAX	Max positive value of a **float**	3.40282346638528860e+38 +
FLT_MIN	Min positive value of a **float**	1.40129846432481707e-45-
INT_MAX	Max decimal value of an **int**	2147483647 +
INT_MIN	Min decimal value of an **int**	-2147483648 -
LINE_MAX	Max number of characters in a single line	2048 =
LINK_MAX	Max number of links to a single file	32767 +*
LOCK_MAX	Max number of entries in system lock table	32 +-*
LONG_BIT	Number of bits in a **long**	32 +
LONG_MAX	Max decimal value of a **long**	2147483647 +
LONG_MIN	Min decimal value of a **long**	-2147483648 -
MAX_CANON	Max number of bytes in terminal canonincal input line	512 +*

MAX_CHAR	Max number of bytes in terminal input queue	MAX_INPUT =*
MAX_INPUT	Max number of bytes in terminal input queue	512 +*
NAME_MAX	Max number of bytes in a path name component	14 +*
NL_ARGMAX	Max value of "digits" in calls to the NLS *printf*(3S) and *scanf*(3S) functions	9 =
NL_MSGMAX	Max message number in an NLS message catalog	32767 +
NL_SETMAX	Max set number in an NLS message catalog	255 +
NL_TEXTMAX	Max number of bytes in an NLS message string	8192 +
NGROUPS_MAX	Max number of supplementary groups per process	20 +
OPEN_MAX	Max number of files a process can have open	60 +*
PASS_MAX	Max number of chars in a password	8 +
PATH_MAX	Max number of characters in a path name excluding the null terminator	1023 +*
PID_MAX	Max value for a process ID	30000 +
PIPE_BUF	Max number of bytes atomic in write to a pipe	8192 +*
PIPE_MAX	Max number of bytes writable to a pipe in one write	INT_MAX +
PROC_MAX	Max number of simultaneous processes on system	84 +-*
SCHAR_MAX	Max integer value of a **signed char**	127 =
SCHAR_MIN	Min integer value of a **signed char**	-128 =
SHRT_MAX	Max decimal value of a **short**	32767 +
SHRT_MIN	Min decimal value of a **short**	-32768 -
STD_BLK	Number of bytes in a physical I/O block	512 +
SYSPID_MAX	Max process ID of system processes	4 +-*
SYS_NMLN	Length of strings returned by *uname*(2)	8 +*
SYS_OPEN	Max number of files open on system	120 +-*
TMP_MAX	Max number of unique names generated by *tmpnam*(3S)	17576 +
UCHAR_MAX	Max integer value of an **unsigned char**	255 =
UID_MAX	Smallest unattainable value for a user or group ID	60000 +
UINT_MAX	Max decimal value of an **unsigned int**	4294967295 +
ULONG_MAX	Max decimal value of an **unsigned long**	4294967295 +
USHRT_MAX	Max decimal value of an **unsigned short**	65535 +
USI_MAX	Max decimal value of an **unsigned int**	UINT_MAX =*
WORD_BIT	Number of bits in a "word" (**int**)	32 +

EXAMPLES

UID_MAX has an HP-UX value of **60000 +** , which means that on all HP-UX systems the smallest unattainable value for a user or group ID is at least 60000. A particular system might be capable of supporting more than 60000 user or group IDs, in which case its **<limits.h>** file sets **UID_MAX** to a higher value; however, any application assuming such a higher value is not guaranteed to be portable to all HP-UX systems.

AUTHOR

limits was developed by HP.

SEE ALSO

exec(2), fcntl(2), fork(2), getgroups(2), link(2), lockf(2), open(2), pathconf(2), sysconf(2), uname(2), write(2), printf(3S), scanf(3S), tmpnam(3S), passwd(4), values(5), termio(7).

Series 300/400 and 700
config(1M).

Series 800
uxgen(1M).

STANDARDS CONFORMANCE

<limits.h>: AES, XPG2, XPG3, XPG4, FIPS 151-2, POSIX.1, POSIX.2, ANSI C

NAME

man - macros for formatting entries in this manual

DESCRIPTION

These macros are used by `nroff` (and are usable by `troff`) to determine the layout format of the on-line version of entries found in this and other related reference manuals. These macros are used by the `man` command (see *man*(1)).

The default page size is 8.5×11 inches, with a 6.5×10-inch text area. The `-rs1` option (which is ignored by `nroff`) reduces these dimensions to 6×9 inches and 4.75×8.375 inches, respectively — and reduces the default type size from 10-point to 9-point; the vertical line spacing from 12 points to 10 points. The `-rV2` option can be used to set certain parameters to values appropriate for certain Versatec printers: line length to 82 characters; page length to 84 lines; underlining inhibited. This option should not be confused with the `-Tvp` option of the `man` command, which is available on some UNIX operating systems.

Any *text* argument below can consist of one to six "words". Double quotes (can be used to include blanks in a "word". If *text* is empty, the special treatment is applied to the next line containing text to be printed. For example, `.I` can be used to italicize a whole line, or `.SM` followed by `.B` to make small bold text.

By default, hyphenation is turned off for `nroff`, but remains on for `troff`. Paragraphs are left-justified, ragged-right for `nroff`, and adjusted both left and right for `troff`.

Type font and size are reset to default values before each paragraph and after processing font- and size-setting macros such as `.I`, `.RB`, and `.SM`. Tab stops are neither used nor set by any macro except `.DT` and `.TH`. `.TH` invokes `.DT` (see below).

Default units for indents *in* are ens. When *in* is omitted, the previous indent is used. This remembered indent is set to its default value (7.2 ens in `troff`, 5 ens in `nroff` — corresponding to 0.5 inch in the default page size) by `.TH`, `.P`, `.PP`, and `.RS`, and restored by `.RE`.

`.TH` *t s c n a* — Set the title and entry heading:
- *t* Entry title.
- *s* Section number. *t* is combined with *s* in parentheses to form the top left- and right-hand corners of the page heading.
- *c* Extra commentary such as "Optional Software Required"; placed in parentheses at the center of the bottom line in the two- or three-line page heading space.
- *n* (for "new manual name") Used for other text such as "Series 300/400 Only"; centered between the title and section on the first page heading line.
- *a* (for "alternate entry name") Provided to support alternate naming such as a FORTRAN routine name corresponding to a C function name specified in *t*.

`.SH` *text* — Place section head *text*, e.g., `SYNOPSIS`, here.

`.SS` *text* — Place sub-section head *text* such as `Options` here.

`.C` *text* — Set *text* in `computer` (monospaced) font. Handled the same as bold by `nroff`.

`.B` *text* — Make *text* bold.

`.I` *text* — Make *text* italic.

`.SM` *text* — Make *text* 1 point smaller than default point size.

`.RI` *a b* — Concatenate Roman *a* with italic *b*, and alternate these two fonts for up to six arguments. Similar macros alternate between any two of Roman, italic, and bold:

```
.IR   .RB   .BR   .IB   .BI
```

`.CI` *a b* — Concatenate computer font *a* with italic *b*, and alternate these two fonts for up to six arguments. Similar macros alternate between any two of computer and Roman, italic, and bold:

```
.IC   .CB   .BC   .CR   .RC
```

Computer font printed the same as bold by `nroff`.

`.P` — Begin a paragraph with normal font, point size, and indent. `.PP` is a synonym for `.P`.

`.HP` *in* — Begin paragraph with hanging indent.

`.TP` *in* — Begin indented paragraph with hanging tag. The next line that contains text to be printed is taken as the tag. If the tag does not fit, it is printed on a separate line.

`.IP` *t in* — Same as `.TP` *in* with tag *t*; often used to get an indented paragraph without a tag.

`.RS` *in* — Increase relative indent (initially zero). Indent all output an extra *in* units from the current left margin.

`.RE` *k* — Return to the *k*th relative indent level (initially, *k*=1; *k*=0 is equivalent to *k*=1); if *k* is omitted, return to the most recent lower indent level.

`.PM m` Produces proprietary markings; where *m* may be `P` for `PRIVATE`, or `N` for `NOTICE`, `BP` for `BELL LABORATORIES PROPRIETARY`, or `BR` for `BELL LABORATORIES RESTRICTED`.

`.DT` Restore default tab settings (every 7.2 ens in `troff`, 5 ens in `nroff`).

`.PD v` Set the interparagraph distance to *v* vertical spaces. If *v* is omitted, set the interparagraph distance to the default value (0.4v in `troff`, 1v in `nroff`).

The following *strings* are defined:

`\*R` `(Reg.)` in `nroff`, Registered Trademark symbol in `troff`, if available.

`\*S` Change to default type size.

`\*(Tm` Trademark indicator.

The following *number registers* are given default values by `.TH`:

`IN` Left margin indent relative to section heads (default is 7.2 ens in `troff`, 5 ens in `nroff`).

`LL` Line length including `IN`.

`PD` Current interparagraph distance.

Font Conventions

Entries in the *HP-UX Reference* use the following font conventions:

Roman Normal typeface used for explanatory and other normal text.

`computer` Used for all literals which are typed exactly as shown when used as keyboard commands or command-line options, in programs, etc.

italic Used for variables and other words that represent an argument that may take on a user-defined or variable value. Also used for *emphasis* in regular text.

boldface Used primarily in headings and occasionally for terms when first introduced or when being defined.

Special Features

Strings used in the page footer macro are initialized by the `.TH` macro. One is defined as a non-printing (null) string to prevent the printing of "Hewlett-Packard Company" at the bottom of manual entries provided by other parties.

This arrangement enables users and third-party software suppliers to directly control the contents of the left- and right-hand fields of the footer line for use in displaying company name, release version, etc., as desired when creating their own manual entries. Footer string `)H` is printed on the left; string `]W` is printed on the right, and the page number is printed in the center. Strings can be defined anywhere after the .TH macro call provided they appear before the end of the first page as in the following example source file segment:

```
.TH man 5
.ds )H XYZ Company
.ds ]W Release 2.3: July 1989
```

which produces a footer resembling:

XYZ Company **- 1 -** **Release 2.3: July 1989**

FILES

```
/usr/lib/macros/cmp.[nt].[dt].an
/usr/lib/tmac/tmac.an
/usr/lib/macros/ucmp.[nt].an
```

SEE ALSO

man(1), nroff(1).

WARNINGS

In addition to the macros, strings, and number registers mentioned above, a number of *internal* macros, strings, and number registers are defined. Except for names predefined by `nroff/troff` and number registers `d`, `m`, and `y`, all such internal names are of the form *XA*, where *X* is one of `)`, `]`, and `}`, and *A* stands for any alphanumeric character.

The **NAME** section of each entry is assumed to consist of a single line of input that has the following format:

name[, name, name ...] \- explanatory text

The **NAME** section is no longer used to prepare the Table of Contents and Index for this manual. Instead, that information is coded as comments at the end of each manual entry source file where it can be accessed by various tools and programs as desired. However, *catman*(1M) and related programs still use the **NAME** line to create the mkwhatis database.

The macro package increases the inter-word spaces (to eliminate ambiguity) in the `SYNOPSIS` section of each entry.

The macro package itself uses only the Roman font (so that one can replace, for example, the bold font by the constant-width font if available). Of course, if the input text of an entry contains requests for other fonts (e.g., `.I`, `.RB`, `\fI`), the corresponding fonts must be mounted. The computer font macros use font position 3 (same as bold). To use a constant-width font, change the font 3 specification in each macro to font 4 and mount the constant-width font in position 4 using a troff `.fp` request.

Any argument to `.TH` containing blanks must be enclosed by double quotes (to ensure correct interpretation for formatting.

NAME

manuals - Current list of HP-UX documentation

DESCRIPTION

This entry contains a list of current English language manuals for the HP-UX operating system. Information is current as of the publication date for this manual. Applications software manuals are not included in this list. See Ordering Information below for information about manuals in other languages.

If discrepancies are encountered between system behavior and a given manual, verify the edition date to ensure that the manual is current.

Manual List

Stock Number	Title
All Series:	
32069-60002	XTI Programmers Guide
32069-60002	XTI Programmers Guide
35328-90002	COBOL/HP-UX Operating Manual
35328-90003	COBOL/HP-UX Utilities Manual
35328-90012	COBOL/HP-UX Pocket Guide
36940-90017	X.25 Programmer's Guide
36940-90018	Install/Administer X.25/9000
36940-90019	Troubleshooting X.25/9000
5010-7168	Mastering Motif Widgets
92501-90001	USL C++ Manual
92501-90001	USL C++ Manual
92501-90005	HP C++ Programmer's Guide
92501-90007	C++ Primer
92501-90014	The Annotated C++ Reference Ma
92552-90009	NLIO Input Method Guide (Trad. Chinese)
97005-90015	The Ultimate Guide to the vi and ex Text Editors
98194-60530	Install. and Admin. LAN/9000
98194-60531	Berkeley IPC Programmer's Guid
98194-60532	NetIPC Programmer's Guide
98194-60533	Using Serial Line IP Protocols
98194-60533	Using Serial Line Protocols
98194-60534	LLA Programmer's Guide
98592-90031	Starbase C Pocket Reference
98592-90041	Starbase FORTRAN Pocket Refere
98592-90047	A Beginner's Guide to Using St
98592-90051	Starbase Pascal Pocket Referen
98592-90080	Starbase Graphics Techniques
98592-90093	Fast Alpha/Font Manager Progra
98672-90040	HP-GKS FORTRAN Pocket Referenc
98672-90602	ANSI Standard - GKS Reference
B1012-90013	Networking Overview
B1012-90014	Installing and Administering NS
B1012-90015	Using Network Services
B1013-90008	Using NFS Services
B1013-90009	Installing and Administering NFS
B1013-90010	Programming & Protocols for NFS
B1014-90008	Administering ARPA Services
B1014-90009	Using ARPA Services
B1171-90021	HP VUE Installation Guide
B1171-90024	P VUE Programmer's Guide
B1171-90026	XLIB Programming and Reference Guide
B1171-90027	X Toolkit Intrinsics Programming Guide
B1171-90028	X Toolkit Intrinsics Reference
B1171-90029	X Window System C Quick Reference
B1171-90030	Mastering Motif Widgets
B1171-90031	Motif 1.1 Information Manual
B1171-90032	HP OSF/Motif Style Guide

B1171-90033	HP OSF/Motif Programmer's Reference
B1171-90034	HP OSF/Motif Programmer's Guide
B1171-90042	HP VUE User's Guide
B1760-90608	Read Me Before Installing the
B1862-90000	A Beginner's Guide to HP-UX
B1862-90001	Finding Information
B1862-90003	Master Index
B1862-90004	HP-UX Error Message Catalog
B1862-90012	Mail Systems: User's Guide
B1862-90013	Terminal Control: User's Guide
B1862-90014	Text Formatting: User's Guide
B1862-90015	Number Processing: User's Guide
B1862-90016	Text Processing: User's Guide
B1864-90002	Device I/O: User's Guide
B1864-90009	C Programming Tools
B2200-90019	Japanese NLIO Manual
B2204-90002	NLIO System Admin Guide (Kor.)
B2204-90011	NLIO Access User's Guide (Kor.)
B2208-90011	NLIO Access User's Guide (Trad. Chinese)
B2208-90012	NLIO System Admin Guide (Trad. Chinese)
B2212-90002	NLIO System Admin Guide (Simp. Chinese)
B2212-90003	NLIO Input Method Guide (Simp. Chinese)
B2212-90011	NLIO Access User's Guide (Simp. Chinese)
B2351-90000	Terminal Session Manager: User
B2355-90019	Starbase Device Drivers Librar
B2355-90020	Starbase Reference
B2355-90025	HP-UX Portability Guide
B2355-90026	Programming on HP-UX
B2355-90029	How HP-UX Works: Concepts for System Administrators
B2355-90030	Solving HP-UX Problems
B2355-90031	Creating Product Packages for
B2355-90033	HP-UX Reference (3 vols)
B2355-90034	POSIX Conformance Document
B2355-90036	NLS User's Guide
B2355-90037	Remote Access: User's Guide
B2355-90044	HP-UX Symbolic Debugger User's
B2355-90045	HP-UX System Security
B2355-90046	Shells: User's Guide
B2355-90612	Starbase Tech Addendum for 9.0
B2361-90000	Starbase Display List Programm
B2362-90000	HP-GKS Device Drivers Library
B2362-90001	HP-GKS Users Guide
B2408-90009	HP-UX FORTRAN/9000 Programming Guide
B2408-90010	HP-UX FORTRAN/9000 Programming Reference
B2408-90011	FORTRAN/9000 Release Notes
B2408-90011	HP-UX FORTRAN/9000 Release Notes
B2408-90012	HP-UX Symbolic Debugger Release Notes
B2617-90000	Codelibs Library Reference 2.1
J2157-61001	Installing/Administering FDDI
J2165-61004	Installing/Admin Token Ring
tbd	Managing Clusters of HP 9000 Computers: Sharing HP-UX File Systems

Series 300/400/700:

98885-90002	Starbase Radiosity and Ray Tracing
B1171-90023	HP VUE Advanced User's Guide
B1171-90037	Using the X Window System
B1171-90043	Using the X Window System
B1171-90044	HP VUE Advanced User's Guide
B1685-90601	Read Me Before Installing HP-PHIGS

B1759-90000	HP-PHIGS Graphics Techniques
B1759-90003	A Beginner's Guide to Using HP
B1759-90011	HP-PHIGS FORTRAN Binding Refererence
B1759-90217	HP PHIGS Workstation Configuration
B1759-90608	HP-PHIGS Development Product R
B1760-90000	Introducing Personal Visualize
B1760-90001	Personal 3D Edit User's Guide
B1760-90002	Personal Visualizer User's Guide
B1760-90003	Personal Translator: IGES User Guide
B1760-90004	Personal Translator: I-DEA Solutions Guide
B1760-90005	Personal Visualizer: Install and Config Guide
B2355-90038	Managing Clusters of HP 9000 C
B2355-90042	Finding HP-UX Information Series 300/400/700
B2362-90600	Read Me Before Installing or Updating HP-UX
B2910-90001	Using HP-UX

Series 700/800:

92431-90005	HP Pascal/HP-UX Reference Manual
92431-90006	HP Pascal/HP-UX Programmer Guide
92431-90007	HP Pascal/HP-UX Quick Reference
92432-90001	Assembly Language Reference
92434-90002	HP C Programmer's Guide
92453-90024	HP C/HP-UX Reference Manual
92501-90007	C++ Primer
B2355-90024	HP-UX Floating-Point Guide

Series 300/400 Only:

	HP-UX Release Notes Series 300/400
	Readme HP-UX 9.0 Series 300/400
35328-90040	Software Issue Bulletin for Series 300/400
B1864-90008	HP C Programmer's Guide
B1700-90002	C: A Reference Manual
B1864-90010	HP-UX System Admin Tasks 300/400
B1864-90011	Installing Peripherals 300/400
B1864-90013	Master Index Series 300/400
B1864-90014	HP-UX Assembler and Tools
B1864-90015	Managing Clusters, Series 300/400
B1864-90019	Installing and Updating HP-UX, Series 300/400

Series 700 Only:

	HP-UX 9.0 Release Notes Series 700
	Readme HP-UX 9.0 Series 700
B2355-90027	Using Audio Appl Interface
B2355-90039	Inst/Update HP-UX S700
B2355-90040	HP-UX System Admin Tasks S700
B2355-90041	Installing Peripherals S700
B2355-90043	Master Index Series 700

Series 800 Only:

5959-5273	Owner's Guide to HP9000 8X7S
5961-1612	Support Tools Mgr User's Guide
92453-90023	Managing Disk Mirrors Using DataPair 9000
92668-90005	Managing SwitchOver/UX
97084-90011	DGL Programmer's Manual
97084-90029	AGP/DGL Device Handlers Manual
97085-90013	Advanced Graphics Package Programming
B3108-90001	Finding Information S800
B3108-90002	HP-UX Release Notes S800
B3108-90003	Master Index Series 800

B3108-90004	Installing Peripherals S800
B3108-90005	HP-UX System Administration Tasks, Series 800
B3108-90006	Install/Update HP-UX 9.0 Series 800

Other:

28604-90001	HP OSI Express 802.4 Hardware
32122-90003	Admin. Ref. Manual for HP OSI Express Link

Ordering Information

For information about how to order any of these manuals as well as other HP computer and calculator manuals and supplies, call **HP Direct** toll-free at 1-800-637-7740 in the United States, or contact your nearest HP Sales and Support office outside the U.S.A.

NAME

math - math functions and constants

SYNOPSIS

#include <math.h>

DESCRIPTION

This file contains declarations of all the functions in the Math Library (described in Section (3M)), as well as various functions in the C Library (Section (3C)) that return floating-point values.

It defines the structure and constants used by the *matherr*(3M) error-handling mechanisms, including the following constant used as an error-return value:

HUGE_VAL The maximum non-infinity value of a double-precision floating-point number.

The following mathematical constants are defined for user convenience:

M_E The base of natural logarithms (*e*).

M_LOG2E The base-2 logarithm of *e*.

M_LOG10E The base-10 logarithm of *e*.

M_LN2 The natural logarithm of 2.

M_LN10 The natural logarithm of 10.

M_PI The ratio of the circumference of a circle to its diameter. (There are also several fractions of its reciprocal and its square root.)

M_SQRT2 The positive square root of 2.

M_SQRT1_2 The positive square root of 1/2.

For the definitions of various machine-dependent "constants", see the description of the **<values.h>** header file.

FILES

/usr/include/math.h

SEE ALSO

intro(3), matherr(3M), values(5).

STANDARDS CONFORMANCE

<math.h>: AES, XPG2, XPG3, XPG4

NAME

mknod.h - macros for handling device numbers

SYNOPSIS

#include <sys/mknod.h>

DESCRIPTION

The header **<sys/mknod.h>** defines macros to create and interpret device identification numbers for use with *mknod*(2).

Use of these macros is architecture dependent. See the System Administration Manual for your system for information on how to select major and minor device numbers.

It contains the macro

dev_t makedev(int major, int minor)

which packs the major and minor components into a device identification number suitable for the *dev* argument of *mknod*(2), and the two macros:

int major(dev_t dev)
int minor(dev_t dev)

which extract the major and minor number components, respectively, from a device identification number, *dev*.

The macro MINOR_FORMAT is a *printf*(3S) specification that prints the minor number in the format best suited to the particular implementation; it is used by the long format of *ls*(1) to show the minor numbers for device files.

The base of the number is indicated in the same way as in the C programming language: no leading zero for decimal, leading zero for octal, and leading **0x** for hexadecimal.

SEE ALSO

ls(1), mknod(1M), mknod(2), printf(3S).

NAME

mm - the MM macro package for formatting documents

SYNOPSIS

mm [*options*] [*files*]
nroff -mm [*options*] [*files*]
nroff -cm [*options*] [*files*]

DESCRIPTION

This package provides a formatting capability for a very wide variety of documents. The manner in which a document is typed in and edited is essentially independent of whether the document is to be eventually formatted at a terminal or is to be phototypeset. See the references below for further details.

The **-mm** option causes *nroff*(1) and *troff* to use the non-compacted version of the macro package, while the **-cm** option results in the use of the compacted version, thus speeding up the process of loading the macro package.

FILES

/usr/lib/macros/cmp.n.[dt].m	compacted version of the package
/usr/lib/macros/mmn	non-compacted version of the package
/usr/lib/tmac/tmac.m	pointer to the non-compacted version of the package
/usr/lib/macros/ucmp.n.m	initializers for the compacted version of the package

SEE ALSO

mm(1), nroff(1).

MM - Memorandum Macros tutorial in *Text Formatters User's Guide*.

NAME

memory mapping definitions

SYNOPSIS

```
#include <sys/mman.h>
```

DESCRIPTION

The <sys/mman.h> header defines the following symbolic constants for use with the `madvise()` function:

`MADV_NORMAL`	No further special treatment.
`MADV_RANDOM`	Expect random page references.
`MADV_SEQUENTIAL`	Expect sequential page references.
`MADV_WILLNEED`	Will need these pages.
`MADV_DONTNEED`	Will not need these pages.
`MADV_SPACEAVAIL`	Ensure that resources are reserved.

The following symbolic constants are defined for use with the `mmap()` and `mprotect()` functions:

`PROT_READ`	Region can be read.
`PROT_WRITE`	Region can be written.
`PROT_EXEC`	Region can be executed.
`PROT_NONE`	Region cannot be accessed.

The following symbolic constants are defined for use with the `mmap()` function:

`MAP_FILE`	Map a file.
`MAP_ANONYMOUS`	Map an unnamed memory region.
`MAP_VARIABLE`	Place region at implementation-computed address.
`MAP_FIXED`	Place region at specified address.
`MAP_SHARED`	Share changes made to mapped region.
`MAP_PRIVATE`	Changes to mapped region are private to a process.

The following symbolic constants are defined for use with the `msync()` function:

`MS_SYNC`	Perform synchronous writes.
`MS_ASYNC`	Perform asynchronous writes.
`MS_INVALIDATE`	Invalidate cached pages.

The following symbolic constants are defined for use with the `msem_init()`, `msem_lock()`, and `msem_unlock()` functions:

`MSEM_LOCKED`	Create semaphore in locked state.
`MSEM_UNLOCKED`	Create semaphore in unlocked state.
`MSEM_IF_NOWAIT`	Do not wait if semaphore is locked.
`MSEM_IF_WAITERS`	Do not unlock if semaphore has no waiters.

The `typedef struct msemaphore` is defined for use with the `msem_init()`, `msem_lock()`, `msem_unlock()`, and `msem_remove()` functions.

SEE ALSO

mmap(2), munmap(2), mprotect(2), msync(2), madvise(2), msem_init(2), msem_remove(2), msem_lock(2), msem_unlock(2).

STANDARDS CONFORMANCE

NAME

ndir.h - format of HP-UX directory streams

SYNOPSIS

#include <ndir.h>

DESCRIPTION

This header file defines data types used by the directory stream routines described in *directory*(3C). It is provided to allow older HP-UX programs to compile unmodified. The header file **<dirent.h>** described on *dirent*(5) should be used in all new programs for compatibility with System V Release 3, the *X/Open Portability Guide*, and the IEEE P1003.1 POSIX standard.

The following data types are defined:

DIR A structure containing information about an open directory stream.

struct direct A structure defining the format of entries returned by the old HP-UX *readdir* function (see *directory*(3C)).

The **struct direct** structure includes the following members:

```
char d_name[MAXNAMLEN+1];   /* name of directory entry */
long d_ino;                 /* file serial number */
short d_namlen;             /* length of string in d_name */
short d_reclen;             /* length of this record */
```

The constant **MAXNAMLEN** is defined in **<ndir.h>**.

This file also contains external declarations for the functions in the *directory*(3C) package, including the following declaration:

extern struct direct *readdir( );

WARNINGS

lint(1) might complain about programs that include this file, although they compile and run correctly.

AUTHOR

ndir.h was developed by the University of California, Berkeley, and HP.

SEE ALSO

directory(3C), dirent(5).

NAME

nlio - Native Language I/O (NLIO) Subsystem

DESCRIPTION

The HP-UX Native Language I/O (NLIO) Subsystem is a set of servers, filters, utilities and libraries that provide means to efficiently input and output multibyte characters on multibyte terminals, printers and the X Window System.

NLIO provides application-transparent multibyte code conversion between the internal code and the external code. Application programs including HP-UX commands can utilize the feature of the multibyte character I/O over multibyte terminals, printers and the X Window System without modifying the I/O portion of the program code or linking any special I/O library.

The supported languages are Japanese, Korean, Simplified Chinese and Traditional Chinese.

NLIO supports the following functionalities:

- Read and write multibyte characters on HP Asian terminals and the X Window System.
- Print multibyte characters with HP Asian or HP LaserJet printers.
- Allow HP-UX commands, such as *cat*(1), *more*(1), and *vi*(1), to use multibyte characters.
- Create and modify user-defined characters.
- Support multibyte characters for the *Starbase Graphics Library*.
- Provide language-specific popular input methods.
- Support code conversion utilities and libraries.
- Create and modify user-defined dictionaries for Japanese and Korean.
- Read and write Japanese multibyte characters on a bitmap display.
- Write Japanese multibyte characters on a bitmap display with the FAFM libraries.
- Provide a set of libraries for Japanese Kana-to-Kanji conversion.

AUTHOR

nlio was developed by HP.

SEE ALSO

Native Language I/O manuals

NAME

portnls - MPE Native Language Support routines

SYNOPSIS

/usr/lib/nls/*

DESCRIPTION

portnls contains a set of library routines that perform miscellaneous language-dependent operations. These routines are also available in the HP MPE operating system.

Localizable programs written in Pascal, FORTRAN, or COBOL, and making use of these routines, can be written and run under HP-UX, and ported to MPE by a mere recompilation, and vice versa.

Each routine is described in an individual manual entry in Section 3X, and can be grouped in the following categories:

Language Information Routines:

nlgetlang	Return current language.
nlinfo	Return language-dependent information (data tables).
nlnumspec	Return information needed for formatting and converting numbers.

Date and Time Routines:

almanac	Return numeric date information for a date in the packed date format returned by the calendar routine.
calendar	Return an MPE calendar date.
clock	Return an MPE clock value.
nlconvclock	Check and convert a time array to an internal format.
nlconvcustdate	Convert a date array to a packed date format.
nlfmtcalendar	Format a packed date using a localized format.
nlfmtclock	Format time of day using a localized format.
nlfmtcustdate	Format a packed date using a custom date.
nlfmtdate	Format a date and time in a localized format.
nlfmtlongcal	Format a packed date using a long calendar format.

Language Formatted-Number Routines:

nlconvnum	Convert a native language formatted number to an ASCII number.
nlfmtnum	Convert an ASCII number to a language-specific formatted number.

Character Array Routines:

nlappend	Append the appropriate language ID to a file name.
nlcollate	Compare two character arrays.
nlfindstr	Search for a array in another array.
nljudge	Judge whether a character is a one-byte or multi-byte Asian character.
nlkeycompare	Determine if a character array is almost equal to another.
nlrepchar	Replace non-displayable characters of a array.
nlscanmove	Move, scan and case shift character arrays.
nlsubstr	Extract a subarray of a array.
nlswitchbuf	Convert a array of characters between phonetic order and screen order.
nltranslate	Translate ASCII arrays to EBCDIC using an conversion table.

Flags:

portnls routines may use flags to select their behavior. Individual manual entries contain symbolic names for the values of those flags. Actual implementation of those values (such as unsigned integer values, octal,

hexadecimal, or bit values) depends on the specific programming language being used.

Some functions are driven by the bit-wise AND or by the bit-wise OR of two of those flags. For example, 0x0051 AND 0x0045 is equal to 0x0041.

The following list contains the hexadecimal values of those constants.

Mask characters for *nlscanmove*(3X) flags

```
M_L     0x0001   /* lowercase */
M_U     0x0002   /* uppercase */
M_N     0x0004   /* numeric */
M_S     0x0008   /* special */
M_WU    0x0010   /* while/until 0/1 */
M_US    0x0020   /* upshift */
M_DS    0x0040   /* downshift */
M_TB    0x0080   /* two byte only flag */
M_OB    0x0100   /* one byte only flag */
```

Masks for *nlsubstr*(3X)

```
F_RETURNERR   0x0000   /* Return an error condition */
F_SPP1        0x0001   /* Start from start position + 1 */
F_SPM1        0x0002   /* Start from start position - 1 */
F_SPBL        0x0003   /* Start from start position. Replace character by blank */
F_SP          0x0004   /* Start from start position regardless value of first character */
F_LMP1        0x0010   /* Move until movelength + 1 is reached */
F_LMM1        0x0020   /* Move until movelength - 1 is reached */
F_LMBL        0x0030   /* Move until movelength is reached. Replace character by blank */
F_LM          0x0040   /* Move until movelength is reached regardless value of last byte */
```

Masks for *nlconvnum*(3X)

```
M_STRIPTHOU      0x0001   /* strip thousands separator */
M_STRIPDEC       0x0002   /* strip decimal separator */
M_NUMBERSONLY    0x0004   /* numbers only in input */
```

Masks for *nlfmtnum*(3X)

```
M_INSTHOU      0x0001   /* insert thousands separator */
M_INSDEC       0x0002   /* insert decimal separator */
M_CURRENCY     0x0004   /* insert currency symbol */
M_LEFTJUST     0x0008   /* left justify */
M_RIGHTJUST    0x0010   /* right justify */
M_RETLENGTH    0x0018   /* left justify and return length */
```

Masks for *nlnumspec*(3X)

```
CURRENCY_PRECEDES   0
CURRENCY_SUCCEEDS   1
CURRENCY_REPLACES   2
```

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

This library is provided for compatibility with the HP MPE operating system. Use the Native Language Support routines for C programmers described in *hpnls*(5) for HP-UX NLS support.

AUTHOR

portnls was developed by HP.

FILES

usr/lib/libportnls.a
usr/lib/nls/$LANG/custdat.cat

SEE ALSO

almanac(3X), calendar(3X), clock(3X), hpnls(5), nlappend(3X), nlconvclock(3X), nlconvcustdate(3X), nlconvnum(3X), nlcollate(3X), nlfmtcalendar(3X), nlfmtclock(3X), nlfmtcustdate(3X), nlfmtdate(3X),

nlfindstr(3X), nlfmtlongcal(3X), nlfmtnum(3X), nlgetlang(3X), nlinfo(3X), nljudge(3X), nlkeycompare(3X), nlnumspec(3X), nlrepchar(3X), nlscanmove(3X), nlsubstr(3X), nlswitchbuf(3X), nltranslate(3X)

MPE Intrinsics Reference Manual.

MPE Native Language Support Reference Manual.

NAME

quota - disk quotas

Remarks

The HP-UX default is to allow *chown*(2). This can interfere with the disk quota mechanism. See the WARNINGS section, below.

DESCRIPTION

Disk quotas can be used by the system administrator to limit the number of files and file blocks owned by a user on a per-file system basis. Only HFS file systems support disk quotas. Separate limits can be established for both the number of files (inodes) and the number of (1-Kbyte) blocks for each user. A *soft* (preferred) and a *hard* limit are established.

For example, user *joe_doe* may have *soft* limits of 1000 blocks and 200 files and *hard* limits of 1200 blocks and 300 files on the root file system (/) containing his $HOME directory and **/tmp**, and soft and hard *block* limits of 100 and 120, respectively, with no explicit *files* limit (0) on the mounted file system **/mnt**.

A time limit is established for each file system which determines how long a user is allowed to exceed the soft limit. The default time limit is one week (7 days).

When a user exceeds his soft limit, a warning is emitted on */dev/tty*. The user can continue to increase utilization over the *soft limit* until he either exceeds the *hard limit* or the established *time limit*. Once either of these events occurs, a message is sent to */dev/tty* and further attempts at file creation and/or increased block utilization will fail. At this point, the user must reduce use of the exceeded limit below the **soft** limit to restore normal operation.

At login time, users exceeding quota limits are reminded (via *login*(1)) of exceeded quotas and appropriate remedial action. The user can check current quota status at any time with the *quota*(1) command.

Quota limits and utilization statistics are maintained by the operating system for each file system for which quotas have been enabled (see *mount*(1M) and *quotaon*(1M)).

Disk quotas are established independently for each user and each file system via *edquota*(1M). This command is also used to establish the limit for the amount of time users are permitted to exceed their soft limit. Default time limit is 1 week.

Limits and usage statistics are stored statically in file *quotas* on the root of each file system for which they are in effect. This file is synchronized with information in the kernel by *quotactl*(2) and whenever an affected file system is unmounted.

Quotas can be enabled automatically at boot (mount) time by adding the **quotas** option to the option list in **/etc/checklist** (see *checklist*(4) and *mount*(1M)). Note that default *mount*(1M) behavior is to *not* enable disk quotas.

Quotas can subsequently be disabled and reenabled with *quotaoff*(1M) and *quotaon*(1M). When disabled, the kernel does not maintain usage statistics and the *quotas* file usage statistics are invalidated by file system activity. Disabling quotas improves performance, but necessitates running *quotacheck*(1M) to update the kernel and *quotas* file after subsequently reenabling quotas.

Users with appropriate privileges can obtain reports of current quota statistics with *repquota*(1M). A somewhat related, but independent command is *quot*(1M), which collects and reports disk utilization independently of the disk quota subsystem.

The *mount*(1M) command reports any file systems for which quotas are enabled.

Data Storage Structure

The **dqblk** data structure (defined in **<quota.h>**), is used by the *quotactl*(2) system call to get or set quota information when the command parameter is set to **Q_GETQUOTA**, **Q_SETQUOTA**, or **Q_SETQLIM**. This structure contains fields that are used to store a user's current file and block count and quota limits for a particular file system.

struct dqblk contains the following members:

```
u_long  dqb_bhardlimit; /* maximum # of disk blocks +1 */
u_long  dqb_bsoftlimit; /* preferred limit on disk blocks */
u_long  dqb_curblocks;  /* current block count */
u_long  dqb_fhardlimit; /* maximum # allocated files +1 */
u_long  dqb_fsoftlimit; /* preferred file limit */
u_long  dqb_curfiles;   /* current # allocated files */
u_long  dqb_btimelimit; /* time limit for excessive block use */
u_long  dqb_ftimelimit; /* time limit for excessive files */
```

NETWORKING FEATURES

quota(1) is able to report quota statistics on remote NFS file systems for which disk quotas are in effect if the remote system provides the RPC *rquotad* service.

rquotad(1M) is provided to allow reciprocal support to other systems.

EXAMPLES

Initial Setup: (performed by user with appropriate privileges)

The kernel must be reconfigured to support disk quotas; see the *System Administrator* manuals. Eligible file systems for disk quota enforcement are of type **hfs** with mount options **rw** and **quota**, as described in *mount*(1M) and *checklist*(4).

For each file system for which quotas are to be enabled, perform the following tasks:

1. Mount the file system.
2. Add **quota** to the existing options list in **/etc/checklist**. For example, change the string **default** for the root (/) entry to **default,quota**.
3. Create the **quotas** file at the root of the file system. For example, for the **/mnt** file system, run the command

   cpset /dev/null /mnt/quotas 600 root bin
4. Establish one or more *prototype* user quotas using *edquota*(1M). If you want all users on your system to have the same limits as *proto_user*, use *edquota* to set those quotas for *proto_user*, then use the **-p** option to *edquota* to replicate those limits for all other users.
5. Run *quotacheck* on the file system to record the current usage statistics.

Adding a new user

To add a new user to the quota system:

1. Use *edquota*(1M) to copy the quotas of an existing user.
2. Run *quotacheck*(1M).

Adding a new file system to an established system

Repeat steps 1 through 5 above under *Initial Setup* for the new file system.

WARNINGS

Quotas can be defeated if the *chown(1)* command or the *chown(2)* system call is accessible to a user. The *setprivgrp(1M)* command can be used to limit access to the *chown(2)* system call so that only a specified group of users are permitted to use the *chown(1)* command or the *chown(2)* system call.

The *sam*(1M) command does not yet support disk quotas. When adding new users or file systems, any desired quotas must be established outside of *sam*.

HP has added features to the original implementation to ensure correctness of the content of the quotas file when quotas are enabled by *mount*(1M) and disabled by *umount*(1M), thus eliminating the need to run *quotacheck*(1M). These features are ineffective, however, if *quotaoff*(1M) and *quotaon*(1M) are used to control quotas.

quotacheck should only be run on a dormant file system to ensure accurate usage information. The *-qv* options to *fsclean*(1M) report on the the current viability of the quotas information.

AUTHOR

Disk Quotas were developed by the University of California, Berkeley, Sun Microsystems, and HP.

FILES

/etc/checklist	default file systems
/etc/mnttab	information on mounted file systems
directory/quotas	quota statistics static storage for file system. Where *directory* is the root of the file system, as specified to *mount*(1M).

SEE ALSO

quota(1), quotactl(2) vfsmount(2), edquota(1M), mount(1M), quot(1M), quotacheck(1M), quotaon(1M), rquotad(1M), setprivgrp(1M)

NAME

rcsintro - description of RCS commands

DESCRIPTION

Revision Control System (RCS) automates the storing, retrieval, logging, identification, and merging of revisions of ASCII text files. RCS is useful for managing files that are revised frequently.

Functions of RCS

- Storage and retrieval of revisions of text files. RCS saves revisions in a space efficient way. Revisions can be retrieved by ranges of revision numbers, symbolic names, dates, authors, and states.
- Maintenance of a complete history of changes. RCS logs all changes automatically. In addition to the text of each revision, RCS stores the author, date and time of check in, and a log message summarizing the change.
- Resolution of access conflicts. When two or more people try to modify the same revision of a file, RCS alerts them and prevents one modification from corrupting the other.
- Maintenance of a tree of revisions. RCS can maintain separate lines of development for each file. It stores a tree structure that represents the ancestral relationships among revisions.
- Merging of revisions and resolution of conflicts. Two separate lines of development of a file can be coalesced by merging. If the revisions to be merged affect the same lines of a file, RCS flags the overlapping changes.
- Release and configuration control. Revisions can be assigned symbolic names and marked as released, stable, experimental, etc. With these facilities, configurations of a file can be described simply and directly.
- Automatic identification of each revision with filename, revision number, creation time, author, etc. This identification is like a stamp that can be embedded at an appropriate place in the text of a revision. These stamps make it simple to determine which revisions of which files make up a given configuration.
- Minimization of secondary storage. RCS uses very little extra space for revisions (only the differences are stored). If intermediate revisions are deleted, the remaining deltas are compressed accordingly.

Getting Started with RCS

The basic user interface is extremely simple. The novice only needs to learn two commands: *ci*(1) and *co*(1). *ci*, short for "check in," deposits the contents of a text file into an archival file called an RCS file. An RCS file contains all revisions of a particular text file. *co*, short for "check out", retrieves revisions from an RCS file.

Suppose you have a file **f.c** that you wish to put under control of RCS. Invoke the check in command:

ci f.c

This command creates the RCS file **f.c,v**, stores **f.c** into it as revision **1.1**, and deletes **f.c**. It also asks you for a description. The description should be a synopsis of the contents of the file. All subsequent check-in commands will ask for a log entry, which should summarize the changes that were made.

Files with names ending with ",**v**" are called RCS files ("v" stands for "versions"), all other files are presumed to be working files. To get back the working file **f.c** in the previous example, use the check out command:

co f.c

This command extracts the latest revision from **f.c,v** and writes it into **f.c**. You can now edit **f.c** and check it back in by invoking:

ci f.c

ci increments the revision number properly. If *ci* complains with the message:

ci error: no lock set by

your system administrator has decided to create all RCS files with the locking attribute set to "strict". In this case, you should have locked the revision during the previous check out. Your last check out should have been:

co -l f.c

Of course, it is too late now to do the check out with locking, because you probably modified **f.c** already, and a second check out would overwrite your modifications. Instead, invoke:

rcs -l f.c

This command will lock the latest revision for you, unless somebody else has already locked it. In that case, you will have to negotiate with that person.

Locking assures that you, and only you, can check in the next update, and avoids nasty problems if several people work on the same file. Even if a revision is locked, it can still be checked out for reading, compiling, etc. All that locking prevents is a check in by anybody but the locker.

If your RCS file is private, i.e., if you are the only person who is going to deposit revisions into it, strict locking is not needed and you can turn it off. If strict locking is turned off, the owner of the RCS file need not have a lock for check in; all others still do. Turning strict locking off and on is done with the commands:

rcs -U f.c

and

rcs -L f.c

If you do not want to clutter your working directory with RCS files, create a subdirectory called RCS in your working directory, and move all your RCS files there. RCS commands will search that directory to find needed files. All the commands discussed above will still work without any modification.

To avoid the deletion of the working file during check in (in case you want to continue editing), invoke:

ci -l f.c

or

ci -u f.c

These commands check in **f.c** as usual, but perform an implicit check out. The first form also locks the checked in revision, the second one does not. Thus, these options save you one check out operation. The first form is useful if locking is strict; the second one if not strict. Both update the identification markers in your working file (see below).

You can give *ci* the number you want assigned to a checked in revision. Assume all your revisions were numbered 1.1, 1.2, 1.3, etc., and you would like to start release 2. The command:

ci -r2 f.c

or

ci -r2.1 f.c

assigns the number 2.1 to the new revision. From then on, *ci* will number the subsequent revisions with 2.2, 2.3, etc. The corresponding *co* commands:

co -r2 f.c

and

co -r2.1 f.c

retrieve the latest revision numbered 2.x and the revision 2.1, respectively. *co* without a revision number selects the latest revision on the "trunk"; i.e., the highest revision with a number consisting of 2 fields. Numbers with more than 2 fields are needed for branches. For example, to start a branch at revision 1.3, invoke:

ci -r1.3.1f.c

This command starts a branch numbered 1 at revision 1.3, and assigns the number 1.3.1.1 to the new revision. For more information about branches, see *rcsfile*(4).

RCS File Naming and Location

RCS recognizes two kinds of files: RCS files (revision archives), and working files. Working filenames are defined by the RCS user, RCS file names are generated by RCS by appending **",v"** to the working file name. Pairs of RCS files and working files can be specified in 3 ways:

- Both the RCS file and the working file are given. The RCS filename is of the form **path1/workfile,v** and the working filename is of the form **path2/workfile**, where **path1** and **path2** are (possibly different or empty) paths and **workfile** is a filename.
- Only the RCS file is given. Then the working file is assumed to be in the current directory and its name is derived from the name of the RCS file by removing **path1/** and the suffix ",v".
- Only the working file is given. Then the name of the RCS file is derived from the name of the working file by removing **path2/** and appending the suffix ",v".

If the RCS filename is omitted or specified without a path, RCS commands look for the RCS file in the directory **./RCS** (or the directory it points to if it is a directory link), then in the current working directory.

RCS Directory Links

RCS supports directory links. If a regular file named RCS exists in the current working directory, RCS interprets the first line as a path name to the directory where RCS files are stored. RCS can follow a chain of up to ten directory links to reach the RCS directory.

Automatic Identification

RCS can put special strings for identification into your source and object code. To obtain such identification, place the marker:

$Header$

into your text, for instance inside a comment. RCS replaces this marker with a string of the form:

$Header: *filename revision_number date time author state*$

With such a marker on the first page of each module, you can always see with which revision you are working. RCS keeps the markers up-to-date automatically. To propagate the markers into your object code, simply put them into literal character strings. In C, this is done as follows:

static char rcsid[] = $Header$;

The command *ident* extracts such markers from any file, even object code and dumps. Thus, *ident* lets you find out which revisions of which modules were used in a given program.

You may also find it useful to put the marker **Log** into your text, inside a comment. This marker accumulates the log messages that are requested during check in. Thus, you can maintain the complete history of your file directly inside it. There are several additional identification markers. See *co*(1) for details.

WARNINGS

Names of RCS files are generated by appending ,v to the end of the working file name. If the resulting RCS file name is too long for the file system on which the RCS file should reside, the RCS command terminates with an error message.

RCS is designed to be used with TEXT files only. Attempting to use RCS with non-text (binary) files will result in data corruption.

AUTHOR

rcsintro was developed by Walter F. Tichy, Purdue University, West Lafayette, IN 47907.
Revision Number: 3.0; Release Date: 83/05/11.

SEE ALSO

ci(1), co(1), ident(1), merge(1), rcs(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(4).

Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

NAME

regexp - regular expression and pattern matching notation definitions

DESCRIPTION

A *regular expression* is a mechanism supported by many utilities for locating and manipulating patterns in text. *pattern matching notation* is used by shells and other utilities for file name expansion. This manual entry defines two forms of regular expressions: *Basic Regular Expressions* and *Extended Regular Expressions ;* and the one form of *Pattern Matching Notation* .

BASIC REGULAR EXPRESSIONS

Basic regular expression (RE) notation and construction rules apply to utilities defined as using basic REs. Any exceptions to the following rules are noted in the descriptions of the specific utilities that use REs.

REs Matching a Single Character

The following REs match a single character or a single collating element:

Ordinary Characters

An ordinary character is an RE that matches itself. An ordinary character is any character in the supported character set except <newline> and the regular expression special characters listed in Special Characters below. An ordinary character preceded by a backslash (\ ) is treated as the ordinary character itself, except when the character is (,), {, or }, or the digits **1** through **9** (see REs Matching Multiple Characters). Matching is based on the bit pattern used for encoding the character; not on the graphic representation of the character.

Special Characters

A regular expression special character preceded by a backslash is a regular expression that matches the special character itself. When not preceded by a backslash, such characters have special meaning in the specification of REs. Regular expression special characters and the contexts in which they have special meaning are:

. [\	The period, left square bracket, and backslash are special except when used in a bracket expression (see RE Bracket Expression).
*	The asterisk is special except when used in a bracket expression, as the first character of a regular expression, or as the first character following the character pair \((see REs Matching Multiple Characters).
^	The circumflex is special when used as the first character of an entire RE (see Expression Anchoring) or as the first character of a bracket expression.
$	The dollar sign is special when used as the last character of an entire RE (see Expression Anchoring).
delimiter	Any character used to bound (i.e., delimit) an entire RE is special for that RE.

Period

A period (.), when used outside of a bracket expression, is an RE that matches any printable or nonprintable character except <newline>.

RE Bracket Expression

A bracket expression enclosed in square brackets ([ ]) is an RE that matches a single collating element contained in the nonempty set of collating elements represented by the bracket expression.

The following rules apply to bracket expressions:

bracket expression

A bracket expression is either a *matching list expression* or a *non-matching list expression*, and consists of one or more expressions in any order. Expressions can be: collating elements, collating symbols, noncollating characters, equivalence classes, range expressions, or character classes. The right bracket (]) loses its special meaning and represents itself in a bracket expression if it occurs first in the list (after an initial ^, if any). Otherwise, it terminates the bracket expression (unless it is the ending right bracket for a valid collating symbol, equivalence class, or character class, or it is the collating element within a collating symbol or equivalence class expression). The special characters

. * [\

(period, asterisk, left bracket, and backslash) lose their special meaning within a bracket expression.

matching list A matching list expression specifies a list that matches any one of the characters represented in the list. The first character in the list cannot be the circumflex. For example, **[abc]** is an RE that matches any of **a**, **b**, or **c**.

non-matching list
A *non-matching list* expression begins with a circumflex (**^**), and specifies a list that matches any character *except* <newline> and the characters represented in the list. For example, **[^abc]** is an RE that matches any character except <newline> or **a**, **b**, or **c**. The circumflex has this special meaning *only* when it occurs first in the list, immediately following the left square bracket.

collating element
A *collating element* is a sequence of one or more characters that represents a single element in the collating sequence as identified via the most current setting of the locale category LC_COLLATE (see *setlocale*(3C)).

collating symbol
A *collating symbol* is a collating element enclosed within bracket-period (**[. . .]**) delimiters. Multi-character collating elements must be represented as collating symbols to distinguish them from single-character collating elements. For example, if the string **ch** is a valid collating element, then **[.ch.]** is treated as an element matching the same string of characters, while **ch** is treated as a simple list of the characters **c** and **h**. If the string within the bracket-period delimiters is not a valid collating element in the current collating sequence definition, the symbol is treated as an invalid expression.

noncollating character
A *noncollating character* is a character that is ignored for collating purposes. By definition, such characters cannot participate in equivalence classes or range expressions.

equivalence class
An *equivalence class* expression represents the set of collating elements belonging to an equivalence class. It is expressed by enclosing any one of the collating elements in the equivalence class within bracket-equal (**[= . . . =]**) delimiters. For example, if **a**, **á**, and **A** belong to the same equivalence class, then **[[=a=]b]**, **[[=á=]b]**, and **[[=A=]b]** are each equivalent to **[aáAb]**.

range expression
A *range expression* represents the set of collating elements that fall between two elements in the current collation sequence as defined via the most current setting of the locale category LC_COLLATE (see *setlocale*(3C)). It is expressed as the starting point and the ending point separated by a hyphen (**-**).

The starting range point and the ending range point must be a collating element, collating symbol, or equivalence class expression. An *equivalence class expression* used as an end point of a range expression is interpreted such that all collating elements within the equivalence class are included in the range. For example, if the collating order is **A**, **a**, **B**, **b**, **C**, **c**, **ch**, **D**, **d**; and **A** and **a** constitute an equivalence class, then the expression **[[=a=]-D]** is treated as **[AaBbCc[.ch.]D]**.

Both starting and ending range points must be valid collating elements, collating symbols, or equivalence class expressions, and the ending range point must collate equal to or higher than the starting range point; otherwise the expression is invalid. For example, with the above collating order and assuming that **E** is a noncollating character, then both the expressions **[[=A=]-E]** and **[d-a]** are invalid.

An ending range point can also be the starting range point in a subsequent range expression. Each such range expression is evaluated separately. For example, the bracket expression **[a-m-o]** is treated as **[a-mm-o]**.

The hyphen character is treated as itself if it occurs first (after an initial **^**, if any) or last in the list, or as the rightmost symbol in a range expression. As examples, the expressions **[-ac]** and **[ac-]** are equivalent and match any of the characters **a**, **c**, or **-**; the expressions **[^-ac]** and **[^ac-]** are equivalent and match any characters except <newline>, **a**, **c**, or **-**; the expression **[%--]** matches any of the characters in the defined collating sequence between **%** and **-** inclusive; the expression **[--@]** matches any of the characters in the defined collating sequence between **-** and **@** inclusive; and the expression **[a--@]** is invalid, assuming **-** precedes **a** in the collating sequence.

character class

A character class expression represents the set of characters belonging to a character class, as defined via the most current setting of the locale category LC_CTYPE. It is expressed as a character class name enclosed within bracket-colon ([: :]) delimiters.

Valid character class expressions and the class they represent are:

[:alpha:]	letters
[:upper:]	upper-case letters
[:lower:]	lower-case letters
[:digit:]	decimal digits
[:xdigit:]	hexadecimal digits
[:alnum:]	letters or decimal digits
[:space:]	characters producing white-space in displayed text
[:print:]	printing characters
[:punct:]	punctuation characters
[:graph:]	characters with a visible representation
[:cntrl:]	control characters

REs Matching Multiple Characters

The following rules may be used to construct REs matching multiple characters from REs matching a single character:

RERE The concatenation of REs is an RE that matches the first encountered concatenation of the strings matched by each component of the RE. For example, the RE **bc** matches the second and third characters of the string **abcdefabcdef**.

*RE** An RE matching a single character followed by an asterisk (*) is an RE that matches zero or more occurrences of the RE preceding the asterisk. The first encountered string that permits a match is chosen, and the matched string will encompass the maximum number of characters permitted by the RE. For example, in the string **abbbcdeabbbbbbcde**, both the RE **b*c** and the RE **bbb*c** are matched by the substring **bbbc** in the second through fifth positions. An asterisk as the first character of an RE loses this special meaning and is treated as itself.

\(*RE*\) A subexpression can be defined within an RE by enclosing it between the character pairs \(and \). Such a subexpression matches whatever it would have matched without the \(and \). Subexpressions can be arbitrarily nested. An asterisk immediately following the \(loses its special meaning and is treated as itself. An asterisk immediately following the \) is treated as an invalid character.

n The expression *n* matches the same string of characters as was matched by a subexpression enclosed between \(and \) preceding the *n*. The character *n* must be a digit from **1** through **9**, specifying the *n*-th subexpression (the one that begins with the *n*-th \(and ends with the corresponding paired \). For example, the expression **^\(.*\)\1$** matches a line consisting of two adjacent appearances of the same string.

If the *n* is followed by an asterisk, it matches zero or more occurrences of the subexpression referred to. For example, the expression **\(ab\(cd\)ef\)Z\2*Z\1** matches the string **abcdefZcdcdZabcdef**.

RE\{*m*,*n*\} An RE matching a single character followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is an RE that matches repeated occurrences of the RE. The values of *m* and *n* must be decimal integers in the range 0 through 255, with *m* specifying the exact or minimum number of occurrences and *n* specifying the maximum number of occurrences. \{*m*\} matches exactly *m* occurrences of the preceding RE, \{*m*,\} matches at least *m* occurrences, and \{*m*,*n*\} matches any number of occurrences between *m* and *n*, inclusive.

The first encountered string that matches the expression is chosen; it will contain as many occurrences of the RE as possible. For example, in the string **abbbbbbbc** the RE **b\{3\}** is matched by characters two through four, the RE **b\{3,\}** is matched by characters two through eight, and the RE **b\{3,5\}c** is matched by characters four through nine.

Expression Anchoring

An RE can be limited to matching strings that begin or end a line (i.e., anchored) according to the following rules:

- A circumflex (**^**) as the **first** RE anchors the expression to the beginning of a line; only strings starting at the first character of a line are matched by the RE. For example, the RE **^ab** matches the string **ab** in the line **abcdef**, but not the same string in the line **cdefab**.
- A dollar sign (**$**) as the last character of an RE anchors the expression to the end of a line; only strings ending at the last character of a line are matched by the RE. For example, the RE **ab$** matches the string **ab** in the line **cdefab**, but not the same string in the line **abcdef**.
- An RE anchored by both **^** and **$** matches only strings that are lines. For example, the RE **^abcdef$** matches only lines consisting of the string **abcdef**.

EXTENDED REGULAR EXPRESSIONS

The extended regular expression (ERE) notation and construction rules apply to utilities defined as using extended REs. Any exceptions to the following rules are noted in the descriptions of the specific utilities using EREs.

EREs Matching a Single Character

The following EREs match a single character or a single collating element:

Ordinary Characters

An ordinary character is an ERE that matches itself. An ordinary character is any character in the supported character set except <newline> and the regular expression special characters listed in Special Characters below. An ordinary character preceded by a backslash (\) is treated as Matching is based on the bit pattern used for encoding the character, not on the graphic representation of the character.

Special Characters

A regular expression special character preceded by a backslash is a regular expression that matches the special character itself. When not preceded by a backslash, such characters have special meaning in the specification of EREs. The extended regular expression special characters and the contexts in which they have their special meaning are:

. [\ ( ) * + ? $ | The period, left square bracket, backslash, left parenthesis, right parenthesis, asterisk, plus sign, question mark, dollar sign, and vertical bar are special except when used in a bracket expression (see ERE Bracket Expression).

^ The circumflex is special except when used in a bracket expression in a non-leading position.

delimiter Any character used to bound (i.e., delimit) an entire ERE is special for that ERE.

Period

A period (.), when used outside of a bracket expression, is an ERE that matches any printable or nonprintable character except <newline>.

ERE Bracket Expression

The syntax and rules for ERE bracket expressions are the same as for RE bracket expressions found above.

EREs Matching Multiple Characters

The following rules may be used to construct EREs matching multiple characters from EREs matching a single character:

RERE A concatenation of EREs matches the first encountered concatenation of the strings matched by each component of the ERE. Such a concatenation of EREs enclosed in parentheses matches whatever the concatenation without the parentheses matches. For example, both the ERE **bc** and the ERE **(bc)** matches the second and third characters of the string **abcdefabcdef**. The longest overall string is matched.

RE+ The special character plus (+), when following an ERE matching a single character, or a concatenation of EREs enclosed in parenthesis, is an ERE that matches one or more occurrences of the ERE preceding the plus sign. The string matched will contain as many occurrences as possible. For example, the ERE **b+c** matches the fourth through seventh characters in the string **acabbbcde**.

*RE** The special character asterisk (*), when following an ERE matching a single character, or a concatenation of EREs enclosed in parenthesis, is an ERE that matches zero or more occurrences of the ERE preceding the asterisk. For example, the ERE **b*c** matches the first character in the string **cabbbcde**. If there is any choice, the longest left-most string that permits a match is chosen. For example, the ERE **b*cd** matches the third through seventh characters in the string **cabbbcdebbbbbbcdbc**.

RE? The special character question mark (?), when following an ERE matching a single character, or a concatenation of EREs enclosed in parenthesis, is an ERE that matches zero or one occurrences of the ERE preceding the question mark. The string matched will contain as many occurrences as possible. For example, the ERE **b?c** matches the second character in the string **acabbbcde**.

Alternation

Two EREs separated by the special character vertical bar (|) matches a string that is matched by either ERE. For example, the ERE **((ab)|c)d** matches the string **abd** and the string **cd**.

Precedence

The order of precedence is as follows, from high to low:

[ ]	square brackets
* + ?	asterisk, plus sign, question mark
^ $	anchoring
	concatenation
|	alternation

For example, the ERE **abba|cde** is interpreted as "match either **abba** or **cde**. It does not mean "match **abb** followed by **a** or **c** followed in turn by **de** (because concatenation has a higher order of precedence than alternation).

Expression Anchoring

An ERE can be limited to matching strings that begin or end a line (i.e., anchored) according to the following rules:

- A circumflex (^) matches the beginning of a line (anchors the expression to the beginning of a line). For example, the ERE **^ab** matches the string **ab** in the line **abcdef**, but not the same string in the line **cdefab**.
- A dollar sign ($) matches the end of a line (anchors the expression to the end of a line). For example, the ERE **ab$** matches the string **ab** in the line cdefab, but not the same string in the line **abcdef**.
- An ERE anchored by both ^ and $ matches only strings that are lines. For example, the ERE **^abcdef$** matches only lines consisting of the string **abcdef**. Only empty lines match the ERE **^$**.

PATTERN MATCHING NOTATION

The following rules apply to pattern matching notation except as noted in the descriptions of the specific utilities using pattern matching.

Patterns Matching a Single Character

The following patterns match a single character or a single collating element:

Ordinary Characters

An ordinary character is a pattern that matches itself. An ordinary character is any character in the supported character set except <newline> and the pattern matching special characters listed in Special Characters below. Matching is based on the bit pattern used for encoding the character, not on the graphic representation of the character.

Special Characters

A pattern matching special character preceded by a backslash (\) is a pattern that matches the special character itself. When not preceded by a backslash, such characters have special meaning in the specification of patterns. The pattern matching special characters and the contexts in which they have their special meaning are:

? * [The question mark, asterisk, and left square bracket are special except when used in a bracket expression (see Pattern Bracket Expression).

Question Mark

A question mark (**?**), when used outside of a bracket expression, is a pattern that matches any printable or nonprintable character except <newline>.

Pattern Bracket Expression

The syntax and rules for pattern bracket expressions are the same as for RE bracket expressions found above with the following exceptions:

The exclamation point character (**!**) replaces the circumflex character (**^**) in its role in a non-matching list in the regular expression notation.

The backslash is used as an escape character within bracket expressions.

Patterns Matching Multiple Characters

The following rules may be used to construct patterns matching multiple characters from patterns matching a single character:

***** The asterisk (*****) is a pattern that matches any string, including the null string.

RERE The concatenation of patterns matching a single character is a valid pattern that matches the concatenation of the single characters or collating elements matched by each of the concatenated patterns. For example, the pattern **a[bc]** matches the string **ab** and **ac**.

The concatenation of one or more patterns matching a single character with one or more asterisks is a valid pattern. In such patterns, each asterisk matches a string of zero or more characters, up to the first character that matches the character following the asterisk in the pattern.

For example, the pattern **a*d** matches the strings **ad**, **abd**, and **abcd**; but not the string **abc**. When an asterisk is the first or last character in a pattern, it matches zero or more characters that precede or follow the characters matched by the remainder of the pattern. For example, the pattern **a*d*** matches the strings **ad**, **abcd**, **abcdef**, **aaaad**, and **adddd**; the pattern ***a*d** matches the strings **ad**, **abcd**, **efabcd**, **aaaad**, and **adddd**.

Rule Qualification for Patterns Used for Filename Expansion

The rules described above for pattern matching are qualified by the following rules when the pattern matching notation is used for filename expansion by *sh*(1), *csh*(1), *ksh*(1), and *make*(1).

If a filename (including the component of a pathname that follows the slash (/) character) begins with a period (.), the period must be explicitly matched by using a period as the first character of the pattern; it cannot be matched by either the asterisk special character, the question mark special character, or a bracket expression. This rule does not apply to *make*(1).

The slash character in a pathname must be explicitly matched by using a slash in the pattern; it cannot be matched by either the asterisk special character, the question mark special character, or a bracket expression. For *make*(1) only the part of the pathname following the last slash character can

be matched by a special character. That is, all special characters preceding the last slash character lose their special meaning.

Specified patterns are matched against existing filenames and pathnames, as appropriate. If the pattern matches any existing filenames or pathnames, the pattern is replaced with those filenames and pathnames, sorted according to the collating sequence in effect. If the pattern does not match any existing filenames or pathnames, the pattern string is left unchanged.

If the pattern ends with a plus (+) sign, the pattern may match either an ordinary file ending in a plus, or a Context Dependent File (CDF). To match a CDF, however, the plus sign must be explicitly matched. It cannot be matched with *, ?, or in a bracket expression.

If the pattern begins with a tilde (~) character, all of the ordinary characters preceding the first slash (or all characters if there is no slash) are treated as a possible login name. If the login name is null (i.e., the pattern contains only the tilde or the tilde is immediately followed by a slash), the tilde is replaced by a pathname of the process's home directory, followed by a slash. Otherwise, the combination of tilde and login name are replaced by a pathname of the home directory associated with the login name, followed by a slash. If the system cannot identify the login name, the result is implementation-defined. This rule does not apply to *sh*(1)*or make*(1).

If the pattern contains a $ character, variable substitution can take place. Environmental variables can be embedded within patterns as:

$*name*

or:

${name}

Braces are used to guarantee that characters following *name* are not interpreted as belonging to *name*. Substitution occurs in the order specified only once; that is, the resulting string is not examined again for new names that occurred because of the substitution.

Rule Qualification for Patterns Used in the case Command

The rules described above for pattern matching are qualified by the following rule when the pattern matching notation is used in the case command of *sh*(1) and *ksh*(1).

Multiple alternative patterns in a single clause can be specified by separating individual patterns with the vertical bar character (|); strings matching any of the patterns separated this way will cause the corresponding command list to be selected.

SEE ALSO

ksh(1), sh(1), fnmatch(3C), glob(3C), regcomp(3C), setlocale(3C), cdf(4), environ(5).

STANDARDS CONFORMANCE

<regexp.h>: AES, SVID2, XPG2, XPG3, XPG4

NAME

signal - Description of signals

DESCRIPTION

HP-UX supports multiple signal interfaces (see *sigaction*(2), *signal*(2), *sigvector*(2), *bsdproc*(2), and *sigset*(2V)) that allow a process to specify the action taken upon receipt of a signal. All supported signal interfaces require specification of a signal, as designated by the **Name** and **Number** shown below. Signal specification can be any of the following except `SIGKILL` or `SIGSTOP`, which cannot be caught or ignored:

Name	Number	Notes	Meaning
`SIGHUP`	01	*A*	hangup
`SIGINT`	02	*A*	interrupt
`SIGQUIT`	03	*A,B*	quit
`SIGILL`	04	*A,B,C*	illegal instruction
`SIGTRAP`	05	*A,B,C*	trace trap
`SIGABRT`	06	*A,B*	software generated abort; see *abort*(3C)
`SIGIOT`	06	*A,B*	software generated signal
`SIGEMT`	07	*A,B*	software generated signal
`SIGFPE`	08	*A,B*	floating point exception
`SIGKILL`	09	*A,D,E,F*	kill
`SIGBUS`	10	*A,B*	bus error
`SIGSEGV`	11	*A,B*	segmentation violation
`SIGSYS`	12	*A,B*	bad argument to system call
`SIGPIPE`	13	*A*	write on a pipe with no one to read it
`SIGALRM`	14	*A*	alarm clock; see *alarm*(2)
`SIGTERM`	15	*A*	software termination signal
`SIGUSR1`	16	*A*	user defined signal 1
`SIGUSR2`	17	*A*	user defined signal 2
`SIGCHLD`	18	*G*	death of a child (see WARNINGS below)
`SIGCLD`	18	*G*	death of a child (see WARNINGS below)
`SIGPWR`	19	*C,G*	power fail (see WARNINGS below)
`SIGVTALRM`	20	*A*	virtual timer alarm; see *getitimer*(2)
`SIGPROF`	21	*A*	profiling timer alarm; see *getitimer*(2)
`SIGIO`	22	*G*	asynchronous I/O signal; see *select*(2)
`SIGWINCH`	23	*G*	window size change; see *termio*(7)
`SIGSTOP`	24	*D,E,H*	stop
`SIGTSTP`	25	*H*	stop signal generated from keyboard
`SIGCONT`	26	*F,G*	continue after stop
`SIGTTIN`	27	*H*	background read attempted from control terminal
`SIGTTOU`	28	*H*	background write attempted to control terminal
`SIGURG`	29	*G*	urgent data arrived on an I/O channel
`SIGLOST`	30	*A*	file lock lost (NFS file locking)

The letters in the **Notes** column in the table above indicate the action taken when the signal is received, and any special conditions on its use:

A The default action is to terminate the process.

B The default action of terminating the process also generates a core image file if possible.

C The action is not reset to SIG_DFL before calling the signal-catching function.

D The signal cannot be ignored.

E The signal cannot be caught.

F The signal will not be held off from a stopped process.

G The default action is to ignore the signal.

H The default action is to stop the process.

All signal interfaces allow specification of an *action* that determines what to do upon the receipt of a signal, and should be one of the following:

`SIG_DFL` Execute the default action, which varies depending on the signal as described above:

A Terminate the receiving process with all of the consequences outlined in *exit*(2).

B If following conditions are met, generate a core image file (see *core*(4)) in the current working directory of the receiving process:

- The effective user ID and the real user ID of the receiving process are equal.
- The effective group ID and the real group ID of the receiving process are equal.
- A regular file named **core** does not exist and can be created, or exists and is writable.

If the file is created, it has the following properties:

- The file mode is 0666, modified by the file creation mode mask (see *umask*(2)).
- The file user ID is equal to the effective user ID of the receiving process.
- The file group ID is equal to the effective group ID of the receiving process.

G Ignore the signal. Do not terminate or stop the receiving process.

H Stop the receiving process. While a process is stopped, any additional signals sent to the process are suspended until the process is restarted (except those marked with Note *F* above, which are processed immediately). However, when the process is restarted, pending signals are processed. When a process that is in an orphaned process group (see *glossary*(9)) receives a `SIGTSTP`, `SIGTTIN`, or `SIGTTOU` signal, the process is not stooped because a process in an orphaned process group is not allowed to stop. Instead, a `SIGHUP` signal is sent to the process, and the `SIGTSTP`, `SIGTTIN`, or `SIGTTOU` is discarded.

`SIG_IGN` Ignore the signal.
When one of the supported signal interface routines is used to set the action of a signal to `SIG_IGN` and an instance of the signal is pending, the pending signal is cleared.

D Signals marked with Note *D* above cannot be ignored.

address Catch the signal.
Upon receipt of the signal, if `signal()` is used to set the action, reset the action for the signal caught to `SIG_DFL` (except signals marked with Note *C*). Then, call the signal-catching function to which *address* points, and resume executing the receiving process at the point where it was interrupted. Signal interface routines other than `signal()` normally do not reset the action for the signal caught. However, `sigaction()` and `sigvector()` provide a way of specifying this behavior (see *sigaction(2)* or *sigvector*(2)).

The signal-catching function is called with the following three parameters:

sig The signal number.

code A word of information usually provided by the hardware.

scp A pointer to the machine-dependent structure *sigcontext* defined in `<signal.h>`.

Depending on the value of *sig*, *code* can be zero and/or *scp* can be NULL. The meanings of *code* and *scp* and the conditions determining when they are other than zero or NULL are implementation-dependent (see DEPENDENCIES below). It is possible for *code* to always be zero, and *scp* to always be NULL.

The pointer *scp* is valid only during the context of the signal-catching function.

Optional parameters can be omitted from the signal-catching function parameter list, in which case the signal-catching function is exactly compatible with UNIX System V. Truly portable software should not use the optional parameters in signal-catching routines.

Upon return from the signal-catching function, the receiving process resumes execution at the point where it was interrupted.

When a signal is caught during the execution of system calls such as **read()**, **write()**, **open()**, or **ioctl()** on a slow device (such as a terminal, but not a file), during a **pause()** system call or a **wait()** system call that does not return immediately because a previously stopped or zombie process already exists, the signal-catching function is executed and the interrupted system call returns a −1 to the calling process with **errno** set to EINTR.

C If the signal is marked with Note *C* above, the action is not reset to **SIG_DFL** before calling the signal-catching function. Furthermore, the action is not reset if any signal interface routine other than **signal()** was used to set the action. See the description of signal catching above.

E If the signal is marked with Note *E* above, the signal cannot be caught.

When any stop signal (**SIGSTOP**, **SIGTSTP**, **SIGTTIN**, **SIGTTOU**) is generated for a process, pending **SIGCONT** signals for that process are discarded. Conversely, when **SIGCONT** is generated for a process, all pending stop signals for that process are discarded. When **SIGCONT** is generated for a stopped process, the process is continued, even if the **SIGCONT** signal is blocked or ignored. If **SIGCONT** is blocked and not ignored, the process remains pending until it is either unblocked or a stop signal is generated.

SIGKILL is sent by the system if an **exec()** system call is unsuccessful and the original program has already been deleted.

WARNINGS

The signals **SIGCLD** and **SIGPWR** behave differently than those described above.

The actions for these signals is modified as follows:

SIGCLD Setting the action for **SIGCLD** to **SIG_IGN** in a parent process prevents exiting children of the calling process from creating a zombie process. If the parent process executes the **wait()** function, the calling process blocks until all of the child processes of the calling processes terminate. The **wait()** function then returns a value of −1 with **errno** set to ECHILD (see *wait*(2)).

If one of the signal interface routines is used to set the action for **SIGCLD** to be caught (that is, a function address is supplied) in a process that currently has terminated (zombie) children, a **SIGCLD** signal is delivered to the parent process immediately. Thus, if the signal-catching function reinstalls itself, the apparent effect is that any **SIGCLD** signals received due to the death of children while the function is executing are queued and the signal-catching function is continually reentered until the queue is empty. Note that the function must reinstall itself after it calls **wait()**, **wait3()**, or **waitpid()**. Otherwise the presence of the child that caused the original signal causes another signal immediately, resulting in infinite recursion.

When processing a pipeline, the Bourne shell (see *sh-bourne*(1)) makes the last process in the pipeline the parent of the preceding processes. Job control shells including C shell, Korn shell and the POSIX shell (see *csh*(1), *ksh*(1), and *sh-posix*(1)) make the shell itself the parent of all processes in the pipeline. Therefore, a process that can receive data from a pipe should not attempt to catch **SIGCLD.**

SIGPWR The **SIGPWR** signal is sent to all processes after a power interruption when power is restored and the system has done all necessary reinitialization. Processes restart by catching (or ignoring) **SIGPWR**.

Applications that wish to recover from power failures should catch **SIGPWR** and take whatever necessary steps to reinitialize itself.

Some implementations do not generate SIGPWR. Only systems with nonvolatile memory can recover from power failures.

DEPENDENCIES

Series 300/400

The SIGPWR signal is not currently generated.

The *code* word is always zero for all signals except SIGILL and SIGFPE. For SIGILL, *code* has the following values:

0 Illegal instruction;
6 Check instruction;
7 TRAPV;
8 Privilege violation.

Refer to the MC680xx processor documentation for more detailed information about the meaning of the SIGILL errors.

For SIGFPE, *code* has the following values:

0 Software floating point exception;
5 Integer divide-by-zero.
0x8xxxxxxx
Any value with the high-order bit set indicates an exception while using the HP98248 floating-point accelerator. The value of (*code* &~ 0x80000000) is the value of the HP98248 status register. Refer to HP98248 documentation for more detailed information.
other any other value indicates an exception while using the MC68881 or MC68882 floating point coprocessor. The value of *code* is the value of the MC68881 or MC68882 status register. Refer to the MC68881 documentation for more detailed information.

Series 700

The signal SIGPWR is not currently generated.

Series 700/800

The structure pointer *scp* is always defined.

The *code* word is always zero for all signals except SIGILL and SIGFPE. For SIGILL, *code* has the following values:

8 Illegal instruction trap;
9 Break instruction trap;
10 Privileged operation trap;
11 Privileged register trap.

For SIGFPE, *code* has the following values:

12 Overflow trap;
13 Conditional trap;
14 Assist exception trap;
22 Assist emulation trap.

Refer to the Series 800 processor documentation provided with your system for more detailed information about the meaning of these errors.

The Instruction Address Offset Queue (program counter) is not advanced when a trap occurs on Series 800 systems. If a signal generated by a hardware trap is masked or has its signal action set to SIG_IGN, the program loops infinitely since the instruction causing the trap is re-executed, causing the trap again. If the signal is received by a signal-catching function in the user program, the instruction that caused the trap is re-executed upon return from the signal-catching function unless program flow is altered by the signal-catching function. For example, the longjmp() routine (see *setjmp*(3C)) can be called. Using longjmp() ensures software portability across different hardware architectures.

AUTHOR

signal was developed by HP, AT&T, and the University of California, Berkeley.

SEE ALSO

kill(1), init(1M), bsdproc(2), exit(2), kill(2), lseek(2), pause(2), sigaction(2), signal(2), sigvector(2), wait(2), sigset(2V), abort(3C), setjmp(3C).

STANDARDS CONFORMANCE

<signal.h>: SVID2, XPG2,

NAME

stat.h - file-specific information

SYNOPSIS

#include <sys/stat.h>

DESCRIPTION

The **<sys/stat.h>** header defines the structure of the data returned by the functions *stat*(), *fstat*(), and *lstat*().

The contents of the *stat* structure include the following members:

```
dev_t    st_dev;      /* ID of device containing a */
                      /* directory entry for this file */
ino_t    st_ino;      /* Inode number */
ushort   st_fstype;   /* Type of filesystem this file  */
                      /* is in; see vfsmount(2)*/
ushort   st_mode;     /* File mode; see mknod(2)*/
short    st_nlink;    /* Number of links */
uid_t    st_uid;      /* User ID of file owner */
gid_t    st_gid;      /* Group ID of file group */
dev_t    st_rdev;     /* Device ID; this entry defined */
                      /* only for char or blk spec files */
cnode_t  st_cnode;    /* Cnode ID of machine */
                      /* where the inode lives */
dev_t    st_realdev;  /* Real device number of device */
                      /* containing the inode for this file */
off_t    st_size;     /* File size (bytes) */
time_t   st_atime;    /* Time of last access */
time_t   st_mtime;    /* Last modification time */
time_t   st_ctime;    /* Last file status change time */
                      /* measured in seconds since */
                      /* 00:00:00 UTC, Jan 1, 1970 */
```

The following symbolic names for the values of the *st_mode* field are defined as indicated:

File type:

```
S_IFMT   0170000 type of file
S_IFSOCK 0140000 socket
S_IFLNK  0120000 symbolic link
S_IFNWK  0110000 network special
S_IFREG  0100000 regular (ordinary)
S_IFBLK  0060000 block special
S_IFDIR  0040000 directory
S_IFCHR  0020000 character special
S_IFIFO  0010000 FIFO special (named pipe)
```

File mode bits:

File miscellaneous mode bits:

```
S_CDF    0004000 directory is a context-dependent file
S_ISUID  0004000 set user id on execution
S_ISGID  0002000 set group id on execution
S_ENFMT  0002000 set file-locking mode to enforced
S_ISVTX  0001000 save swapped text even after use
```

File permission mode bits:

```
S_IRWXU 0000700 owner's file access permission bits
S_IRUSR 0000400 read access permission for owner
S_IWUSR 0000200 write access permission for owner
S_IXUSR 0000100 execute/search access permission for owner
S_IRWXG 0000070 group's file access permission bits
```

S_IRGRP 0000040 read access permission for group
S_IWGRP 0000020 write access permission for group
S_IXGRP 0000010 execute/search access permission for group
S_IRWXO 0000007 others' access permission bits
S_IROTH 0000004 read access permission for others
S_IWOTH 0000002 write access permission for others
S_IXOTH 0000001 execute/search access permission for others

Obsolete names for file permission mode bits:

S_IREAD 0000400 read access permission for owner
S_IWRITE 0000200 write access permission for owner
S_IEXEC 0000100 execute/search access permission for owner

File type test macros:

S_ISBLK(*m*) test for a block special file
S_ISCDF(*m*) test for a context-dependent file
S_ISCHR(*m*) test for a character special file
S_ISDIR(*m*) test for a directory
S_ISFIFO(*m*) test for a FIFO special file
S_ISLNK(*m*) test for a symbolic link
S_ISNWK(*m*) test for a network special
S_ISREG(*m*) test for a regular file
S_ISSOCK(*m*) test for a socket

SEE ALSO

chmod(2), chown(2), link(2), mkdir(2), mkfifo(2), mknod(2), stat(2), symlink(2), umask(2), utime(2), types(5).

STANDARDS CONFORMANCE

<sys/stat.h>: AES, SVID2, XPG2, XPG3, XPG4, FIPS 151-2, POSIX.1

NAME

stdarg.h - macros for handling variable argument lists

SYNOPSIS

#include <stdarg.h>
void va_start(va_list pvar, argN);
type **va_arg(va_list pvar,** *type***);**
void va_end(va_list pvar);

DESCRIPTION

The **<stdarg.h>** header contains a set of macros that can be used to write portable procedures that accept variable argument lists. Routines that have variable argument lists (such as *printf*(3S)) but do not use *stdarg* are inherently nonportable, because different machines use different argument-passing conventions.

va_list is a type defined for the variable used to traverse the list.

va_start is called to initialize *pvar* to the beginning of the list. The type of *argN* should be the same as the argument to the function just before the variable portion of the argument list.

va_arg returns the next argument in the list pointed to by *pvar*. *type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, because it cannot be determined at runtime.

va_end is used to clean up.

Multiple traversals, each bracketed by *va_start* ... *va_end*, are possible.

EXAMPLE

This example is a possible implementation of *execl* (see *exec*(2)):

```
#include <stdarg.h>
#define MAXARGS        100

/*      execl is called by
                execl(file, arg1, arg2, ..., (char *)0);
*/
execl(const char *file, const char *args, ...)
{
   va_list ap;
   char *array[MAXARGS];
   int argno = 0;

        va_start(ap, args);
        if ((array[0] = args) != 0)
           while ((array[argno++] = va_arg(ap, char *)) != 0)
                ;
        va_end(ap);
        return execv(file, array);
}
```

SEE ALSO

exec(2), vprintf(3S), varargs(5).

WARNINGS

It is up to the calling routine to specify how many arguments there are, since it is not always possible to determine this from the stack frame. For example, *execl*() is passed a zero pointer to signal the end of the list, and *printf*() can tell how many arguments are there by the format string.

Unless ANSI C is used, it is non-portable to specify a second argument of *char*, *short*, or *float* to *va_arg*, because arguments seen by the called function are never *char*, *short*, or *float*.

Pre-ANSI C converts *char* and *short* arguments to *int* and converts *float* arguments to *double* before passing them to a function.

STANDARDS CONFORMANCE

<stdarg.h>: AES, XPG4, FIPS 151-2, POSIX.1, ANSI C

va_arg: XPG4, ANSI C

va_end: XPG4, ANSI C
va_list: XPG4, ANSI C

va_start: XPG4, ANSI C

NAME

stdsyms - description of HP-UX header file organization

DESCRIPTION

HP-UX header files are organized in a manner that allows for only a subset of the symbols available in that header file to be visible to an application that conforms to a specific standard. The ANSI C, POSIX.1, POSIX.2, and XPG4 standards each reserve a certain set of symbols for that standard's namespace. In addition, the HP-UX implementation of XPG3 and the OSF AES/OS provides for a clean namespace although this is not a specific requirement of those standards.

The following rules apply in determining what symbols are reserved for any standard. These symbols are reserved for the standard and for use by the implementation, and must be either avoided altogether, or used exactly as defined by the specified standard.

- All symbols defined by the desired standard are reserved. Refer to the appropriate standards documentation for a complete list of reserved symbols.
- All symbols beginning with an underscore followed by another underscore or an uppercase letter are reserved for the implementation.
- All external identifiers beginning with an underscore are reserved for the implementation.

The following is a list of feature test macros which must be defined to obtain the appropriate namespace from the header files.

__STDC__

This symbol is automatically defined by the ANSI C pre-processor (**/lib/cpp.ansi**) and is automatically defined when specifying an ANSI C compile (**cc -Aa**). Using the strict ANSI option **-Aa** requests a pure ANSI C namespace, which is the smallest subset of the HP-UX namespace available. The **-Aa** option also enables the inclusion of ANSI-C-style function prototypes for increased type checking. Note that the default namespace when using the **-Aa** option is the ANSI C namespace; therefore a broader namespace must be *requested* if it is desired.

_POSIX_SOURCE

As documented in the IEEE POSIX.1 standard, the programmer is required to define the **_POSIX_SOURCE** feature test macro to obtain the POSIX.1 namespace and POSIX.1 functionality. This feature test macro can be defined, either by using compiler options (**-D_POSIX_SOURCE**) or by using **#define** directives in the source files before any **#include** directives. Note that the default POSIX namespace is the POSIX.1-1990 namespace. It is necessary to define the **_POSIX1_1988** feature test macro in addition to the **_POSIX_SOURCE** macro in order to obtain the POSIX.1-1988 namespace.

_POSIX_C_SOURCE

As documented in the IEEE POSIX.2 standard, the programmer is required to define the **_POSIX_C_SOURCE** feature test macro with a value of 2 to obtain the POSIX.1 and POSIX.2 namespaces and functionality. This feature test macro can be defined, either by using compiler options (**-D_POSIX_C_SOURCE=2**) or by using **#define** directives in the source files before any **#include** directives. This macro is also automatically defined in the XPG4 X/Open namespace (that is, whenever **_XOPEN_SOURCE** and **_XPG4** are defined without defining **_XPG2** or **_XPG3**).

_XOPEN_SOURCE

As documented in the XPG3 and XPG4 standards, the programmer is required to define the **_XOPEN_SOURCE** feature test macro to obtain X/Open functionality. This feature test macro can be defined, either by using compiler options (**-D_XOPEN_SOURCE**) or by using **#define** directives in the source files before any **#include** directives. Although XPG3 does not specify any namespace pollution rules, XPG4 has instituted such rules. Therefore, the HP-UX operating system provides clean namespaces whenever **_XOPEN_SOURCE** is defined.

The current default X/Open namespace is that corresponding to XPG3. To request other versions of the X/Open namespace, define **_XPG2** or **_XPG4** in conjunction with **_XOPEN_SOURCE**.

_AES_SOURCE
As documented in the OSF AES/OS standard, the programmer is required to define the **_AES_SOURCE** feature test macro to obtain OSF functionality. This feature test macro can be defined, either by using compiler options (**-D_AES_SOURCE**) or by using **#define** directives in the source files before any **#include** directives. Although the AES does not specify any namespace pollution rules, the other standards have instituted such rules. Therefore HP-UX provideds a clean namespace whenever **_AES_SOURCE** is defined.

_HPUX_SOURCE
The programmer can define the **_HPUX_SOURCE** feature test macro to obtain the HP-UX namespace and complete HP-UX functionality. Note that the HP-UX namespace is currently a superset of all of the above mentioned namespaces. When using the compatibility-mode compiler (*cc*(1) without the **-Aa** option) or the C++ compiler, the HP-UX namespace is provided by default. The programmer must request one of the other namespaces as described above to obtain the appropriate subset of the HP-UX namespace. When using the strict ANSI-C-mode compiler (**cc -Aa**), the programmer must specifically request a broader namespace.

The **_HPUX_SOURCE** feature test macro can be defined, either by using compiler options (**-D_HPUX_SOURCE**) or by using **#define** directives in the source files before any **#include** directives.

The following is a list of miscellaneous feature test macros that provide various additional features.

__cplusplus
This symbol is automatically defined by the C++ compiler. Defining this macro enables the ANSI-C-style function prototypes for increased type checking. Note that the C++ standard does not specify a namespace; thus the default namespace for C++ is the HP-UX namespace (**_HPUX_SOURCE**).

_POSIX1_1988
This feature test macro should be defined when the POSIX.1-1988 namespace is required. It should be used in conjunction with the **_POSIX_SOURCE** macro if the default POSIX.1-1990 namespace is not desired.

This macro is defined automatically whenever **_AES_SOURCE** or **_XPG3** is requested.

_XPG2 The **_XPG2** macro can be defined when using the compatibility-mode compiler to obtain XPG2 functionality. This provides XPG2 specified function declarations and macros in the HP-UX namespace. Note that the values obtained from most of the macros available when using this option are now available at run-time via the **pathconf()**, **fpathconf()**, and **sysconf()** system calls (see *pathconf*(2) and *sysconf*(2)). Use of the **_XPG2** macro is strongly discouraged because it gives access to obsolete functionality. Note that no function prototypes are provided when using this feature test macro.

_XPG3 The **_XPG3** feature test macro is defined automatically if the programmer has requested the XPG3 namespace (i.e., defined **_XOPEN_SOURCE**, but not some other conflicting namespace such as **_XPG2** or **_XPG4**).

_XPG4 The **_XPG4** feature test macro is provided so that the programmer can obtain the XPG4 namespace, since it differs slightly from the **_XPG3** namespace. In order to obtain the XPG4 namespace, the programmer must define both the **_XOPEN_SOURCE** and **_XPG4** feature test macros. The **_XOPEN_SOURCE** and **_XPG4** feature test macros can be defined, either by using compiler options (**-D_XOPEN_SOURCE -D_XPG4**) or by using **#define** directives in the source files before any **#include** directives.

_SVID2 The **_SVID2** macro can be defined when using the compatibility mode compiler to obtain SVID2 function return types in the HP-UX namespace. The default return types of many functions have since been changed in the HP-UX operating system to align with the ANSI C, POSIX, X/Open, and OSF standards.

_CLASSIC_TYPES
The **_CLASSIC_TYPES** macro can be defined by the programmer to obtain pre-7.0 style function return types and structure element types. This macro has been provided only as a transition aid when migrating from the pre-7.0 version of HP-UX to standards-based HP-UX.

Use of this macro is strongly discouraged as this functionality will be removed in a future release of HP-UX. Note that no function protoytpes are provided when using this feature test macro.

SEE ALSO

cc(1), cpp(1), pathconf(2), sysconf(2).

NAME

suffix - file-name suffix conventions

DESCRIPTION

The following list summarizes file name suffix conventions that can be found in an HP-UX system. It is a partial compilation of possibly useful knowledge, suggestions, and explanations, rather than a specification of standards. Suffixes are often used in preference to prefixes because they enable related files to group together alphabetically in a directory listing.

Note that some programs require the use of a specific value, or vary their behavior based on a choice of suffixes. Such programs are noted in many (but not all) cases.

.A HP64000 cross assembler symbol file.

.a Library file (archive) managed by *ar*; known to *make*.

.ad? HP Ada source, where "?" stands for any single character.

.allow List of users allowed by *at* or *cron* (for example, at.allow).

.an Source for nroff "man" macros.

.ASC LIF (Logical Interchange Format) type 1, ASCII file for use by Pascal or BASIX/UX. Incompatible with *lifcp*.

.aux Cross-referencing information created automatically by LaTeX.

.awk *awk* script file.

.b Compiled LISP (.l) source file, or a bold font file.

.back
.bak
.bkup Backup copy of a file.

.BAD
.bad File containing bad data, or occupying a bad spot on a disk.

.bbl Bibliography created by BibTeX for inclusion in a LaTeX document.

.bib Bibliographic data file, (for example, BibTeX bibliography database).

.blg Log of errors from BibTeX.

.bst BibTeX bibliography style definition.

.C File compressed by *compact*, or C++ language source file, or HP64000 cross compiled C source file.

.c C language source file; known to *cc* and *make*.

.cas CAST language scripts.

.cat NLS (Native Language Support) message catalog.

.cf Configuration file (for example, sendmail.cf).

.clu CLU file.

.CODE Pascal workstation object code.

.cpio File containing output from *cpio* **-o**, that is, a *cpio* archive.

.csh C-shell (*csh*) script.

.curr Current version of a file.

.d Directory file, or data file.

.day A script that is read daily.

.deny List of users denied by *at* or *cron* (for example, cron.deny).

.devs List of devices.

.diff	Differences between two files, output from *diff*.
.dir	DBM database directory file.
.doc	Documentation file of some sort.
.dvi	Device-independent text formatter output.
.e	Extended FORTRAN language (EFL) source file; known to *make*.
.el	GNU Emacs Elisp file.
.elc	Compiled GNU Emacs Elisp file.
.eqn	Source for *nroff* equation macros.
.err	Standard error from a program.
.errors **.errs**	Errors recorded by a program.
.f	FORTRAN language source file; known to *fc* and *make*.
.f77	FORTRAN 77 language source file.
.fc	Frozen configuration file (for example, sendmail.fc).
.full	A complete file or list.
.gf	TeX font bitmaps in Generic Font format.
.glo	Glossary created by LaTeX.
.h	C language header (include) file; known to *make*.
.help **.hf** **.hlp**	Help text for a program, often read automatically.
.hour **.hr**	A script that is read hourly.
.i	Output of C preprocessor ("cc -P"), or a Berkeley Pascal language include file, or an italicized font file.
.icn	Icon source code.
.idx	Index created by LaTeX.
.in	Standard input to a program.
.INDEX	*notes* index file.
.ksh	Korn shell script file.
.L	HP64000 cross linker symbol file.
.l	*lex* source file (known to *make*), or LISP source file.
.LIST	*notes* list file.
.list	File containing a list of other files.
.ln	Library information for *lint*.
.lof	List of figures created by LaTeX.
.log	Generic log file, or a log of error messages from TeX.
.lot	List of tables created by LaTeX.
.m	Modula language source file.
.m2	Modula-2 language source file.
.make	

.mk Makefile for *make*.

.man Source for *nroff* or *troff* using **man** macros.

.me Source for *nroff* or *troff* using **me** macros.

.mf TeX metafont input file.

.ml Gosling/Unipress Emacs Mock Lisp file.

.mm Source for *nroff* or *troff* using **mm** macros.

.mon
.month A script that is read monthly.

.ms Source for *nroff* or *troff* using **ms** macros.

.n *nroff* source.

.NEW
.new New version of a file.

.nro *nroff* source.

.O HP64000 listing file.

.o Relocatable object file (post-compile, pre-link); known to *as*, *cc*, *fc*, *pc*, and *make*.

.obs Obsolete version of a file.

.OLD
.old Old version of a file.

.opt File containing optional material, such as an optional part of the kernel.

.orig Original version of a file.

.out Standard output (and possibly standard error) from a program (for example, nohup.out), or an executable file output from *ld* (such as a.out).

.P HP64000 cross compiled Pascal source file.

.p Pascal language source file (known to *pc* and *make*), or PROLOG language source file.

.pag DBM database data file.

.pi PILOT language source file.

.pk TeX font bitmaps in Packed Font format; denser/more recent than GF.

.prev Previous version of a file.

.ps PostScript files.

.pxl TeX font bitmaps in uncompressed format; very obsolete.

.R HP64000 relocatable file.

.r RatFor language source file; known to *make*.

.rc A "run commands" file, normally read when a program is invoked (for example, mailx.rc).

.real Real version of a file, often one which was replaced by a front-end (for example, uucico.real).

.req File containing required material, such as a required part of the kernel.

.S HP64000 cross assembled source file.

.s Assembler input file; known to *cc* and *make*.

.safe
.save Safe or saved copy of a file.

.scm Scheme file.

.sh Bourne shell script file; known to *make*.

.shar	Shell archive file containing output from *shar*.
.skel	Skeletal or template file.
.sl	Shared library file built by *ld*(1);*known* to *ld*(1).
.st	File containing statistics (for example, **/usr/lib/sendmail.st**).
.sty	LaTeX style definition; should have a corresponding .doc file.
.SYSTM	LIF Bootable by the Series 300/400 boot ROM (see Librarian chapter of *Pascal 3.2 Workstation System,* vol. 1).
.t	Text file.
.tar	File (archive) containing output from *tar*.
.tbl	Source for *nroff* table macros.
.temp **.tmp**	Temporary file.
.template	Prototype or template file.
.test	Test input or output file.
.tex	TeX source file.
.TEXT	*notes* text file, or a Pascal workstation "UCSD text format" file.
.text **.txt**	ASCII text file.
.tfm	Width information used by TeX (TeX font metrics).
.toc	Source for *nroff* table of contents macros, or table of contents created by LaTeX.
.tro	*troff* source.
.u1 **.u2**	Icon intermediate code files.
.UX	HP-UX text or binary file format.
.web	Web file (Knuth's Web system).
.week **.wk**	A script that is read weekly.
.X	HP64000 absolute file.
.y	*yacc* input file; known to *make*.
.Z	File compressed by *compress*.
.z	File compressed by *pack*.
.1 .. .8	Manual entry files (sections 1 through 8), optionally followed by a letter a..z.
.<date>	File saved on given date (year, month name, YYMM, MMDD, etc.) as a snapshot of a continuously-growing logfile.
,v	RCS delta file; known to the RCS programs.

AUTHOR

suffix was developed by HP.

NAME

term - conventional names for terminals

DESCRIPTION

The environment variable TERM is maintained as part of the shell environment (see *profile*(4), and *environ*(5)) and is used by some commands (for example, *tabs*(1)). The *tset*(1) command can be used to set the TERM variable. The name to which TERM is set usually exists in a compiled *terminfo* database (see *terminfo*(4)). The following names are always available in the *terminfo* database:

hp Minimal subset of the capabilities of all Hewlett-Packard terminals and terminal emulators supported on HP-UX systems. Note that entries for specific models of terminals are generally available, and that they often provide better use of the features of those terminals.

dumb Generic name for terminals that lack reverse line-feed and other special escape sequences.

dialup Generic name for dial-in ports connected to unknown terminals.

The TERM variable is also used by commands that use terminal and printer description files from the **/usr/lib/terminfo** directory (such as *nroff*(1), *man*(1), and *tabs*(1)). One TERM name that has a file in this directory is:

lp Generic name for a line printer.

A basic terminal name can have a maximum of eight characters comprised of A-Z, a-z, 0-9, and -. Terminal submodels and operational modes are distinguished by suffixes beginning with a -. Names should be based on original vendors, rather than local distributors. Terminals acquired from the same vendor should be designated with the same basic name.

Commands whose behavior depends on the type of terminal used should accept arguments of the form **-T***term*, where *term* is one of the names given above. If no such argument is present, these commands should obtain the terminal type from the environment variable **$TERM**, which should contain *term*.

WARNINGS

The TERM variable is used differently by commands that originated from UCB code (such as *vi*(1) and *more*(1)) and commands that originated from Bell System III code (such as *nroff*(1) and *tabs*(1)). These different usages of TERM can be confusing.

The inclusion of other names in the *terminfo* database or the **/usr/lib/term** directory does not imply support of these devices.

AUTHOR

term was developed by AT&T and the University of California, Berkeley.

SEE ALSO

man(1), mm(1), nroff(1), sh(1), stty(1), tabs(1), tset(1), ul(1), curses(3X), profile(4), terminfo(4), ttytype(4), environ(5).

NAME

types - primitive system data types

SYNOPSIS

#include <sys/types.h>

REMARKS

The example given on this page is a typical version; the type names are in general expected to be present, although exceptions can be described in DEPENDENCIES. In most cases the fundamental type which implements each typedef is implementation dependent as long as source code which uses those typedefs need not be changed. In some cases the typedef is actually a shorthand for a commonly used type, and will not vary.

DESCRIPTION

The data types defined in the include file are used in HP-UX system code; some data of these types are accessible to user code:

```
typedef struct { int r[1]; }    *physadr;
typedef long                    daddr_t;
typedef char                    *caddr_t;
typedef unsigned int            uint;
typedef unsigned short          ushort;
typedef ushort                  ino_t;
typedef short                   cnt_t;
typedef long                    time_t;
typedef long                    dev_t;
typedef long                    off_t;
typedef long                    paddr_t;
typedef long                    key_t;
typedef short                   pid_t;
typedef long                    uid_t;
typedef long                    gid_t;
```

Note that the defined names above are standardized, but the actual type to which they are defined may vary between HP-UX implementations.

The meanings of the types are:

physadr used as a pointer to memory; the pointer is aligned to follow hardware-dependent instruction addressing conventions.

daddr_t used for disk addresses except in an inode on disk, see *fs*(4).

caddr_t used as an untyped pointer or a pointer to untyped memory.

uint shorthand for *unsigned integer*.

ushort shorthand for *unsigned short*.

ino_t used to specify I-numbers.

cnt_t used in some implementations to hold reference counts for some kernel data structures.

time_t time encoded in seconds since 00:00:00 GMT, January 1, 1970.

dev_t specifies kind and unit number of a device, encoded in two parts known as major and minor.

off_t offsets measured in bytes from the beginning of a file.

paddr_t used as an integer type which is properly sized to hold a pointer.

key_t the type of a key used to obtain a message queue, semaphore, or shared memory identifier, see *stdipc*(3C).

pid_t used to specify process and process group identifiers.

uid_t used to specify user identifiers.

gid_t user to specify group identifiers.

DEPENDENCIES

HP Clustered Environment

The following additional type is defined:

```
typedef ushort  cnode_t;
```

cnode_t is the cnode ID of a machine in a cluster.

SEE ALSO

fs(4), stdipc(3C).

STANDARDS CONFORMANCE

`<sys/types.h>`: AES, XPG2, XPG3, XPG4, FIPS 151-2, POSIX.1

NAME

unistd.h - standard structures and symbolic constants

SYNOPSIS

```
#include <unistd.h>
```

DESCRIPTION

The header <unistd.h> defines the following structures and symbolic constants:

Symbolic constants for the `access()` function:

R_OK	Test for read permission
W_OK	Test for write permission
X_OK	Test for execute (search) permission
F_OK	Test for existence of file

The constants F_OK, R_OK, W_OK , and X_OK and the expressions R_OK|W_OK, R_OK|X_OK, and R_OK|W_OK|X_OK all have distinct values.

Symbolic constant representing a null pointer:

NULL

Symbolic constants for the `lseek()` and `fcntl()` functions (the following constants have distinct values):

SEEK_SET	Set file offset to "offset"
SEEK_CUR	Set file offset to current plus "offset"
SEEK_END	Set file offset to EOF plus "offset"

Symbolic constants (with fixed values):

_POSIX_VERSION
: Integer value indicating version of *IEEE Std 1003.1* standard implemented. The current value is 199009L, indicating the (4-digit) year and (2-digit) month that the standard was approved by the IEEE Standards Board. However, if any of the symbols _AES_SOURCE, _XPG3, or _POSIX1_1988 is defined before <unistd.h> is included, the value of this symbol will be 198808L.

_POSIX2_VERSION
: Integer value indicating version of *IEEE Std 1003.2* standard implemented. The current value is 199210L, indicating the (4-digit) year and (2-digit) month that the standard was approved by the IEEE Standards Board.

_POSIX2_C_VERSION
: Integer value indicating version of *IEEE Std 1003.2* C-Language Binding Option implemented. The current value is 199210L, indicating the (4-digit) year and (2-digit) month that the standard was approved by the IEEE Standards Board.

_XOPEN_VERSION
: Integer value indicating issue number of the X/Open Portability Guide implemented. The current value is 4, indicating Issue 4. However, if the symbol _XPG3 is defined before <unistd.h> is included, the value of this symbol will be 3.

The following symbolic constants are defined in this header if the state of the corresponding option or restriction does not vary after compilation. If a symbol is absent from this header, the value or presence of the corresponding option or restriction should be determined at execution time through `sysconf()` or `pathconf()`:

_POSIX_CHOWN_RESTRICTED	the use of `chown()` is restricted to processes that have appropriate privileges
_POSIX_JOB_CONTROL	implementation supports job control (true of all HP-UX implementations)
_POSIX_NO_TRUNC	pathname components longer than NAME_MAX generate an error

`_POSIX_SAVED_IDS`	effective user and group are saved across an `exec()` call (true of all HP-UX implementations)
`_POSIX_VDISABLE`	terminal special characters can be disabled using this character (see *termio*(7))
`_POSIX2_C_BIND`	all POSIX.2 C-language functionality is provided in the default libraries used by the `c89` C compiler (see *cc*(1)).
`_POSIX2_LOCALEDEF`	new locales can be defined by using the `localedef` command (see *localedef*(1M)).
`_POSIX2_UPE`	the system supports *IEEE Std 1003.2a* (POSIX User Portability Utilities Option)
`_POSIX2_CHAR_TERM`	at least one terminal exists that supports all required POSIX.2a commands.

All symbolic constants whose names begin `_CS`, `_PC`, and `_SC` (see *confstr*(3C), *pathconf*(2), and *sysconf*(2)) are defined.

The following symbolic constants for file streams are defined:

`STDIN_FILENO`	File number of standard input (*stdin*).
`STDOUT_FILENO`	File number of standard output (*stdout*).
`STDERR_FILENO`	File number of standard error (*stderr*).

The types `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t`, and `pid_t` are defined.

Declarations are provided for the following functions:

access()	fork()	initgroups()	setpgrp()
alarm()	fpathconf()	ioctl()	setpgrp2()
brk()	fsync()	isatty()	setresgid()
chdir()	ftruncate()	link()	setresuid()
chown()	getcdf()	lockf()	setsid()
chroot()	getcontext()	logname()	setuid()
close()	getcwd()	lseek()	setusershell()
confstr()	getegid()	lsync()	sgetl()
crypt()	geteuid()	mkstemp()	sleep()
ctermid()	getgid()	mktemp()	sputl()
cuserid()	getgroups()	nice()	swab()
dup()	gethcwd()	pathconf()	swapon()
dup2()	gethostname()	pause()	symlink()
encrypt()	getlogin()	pipe()	sync()
endusershell()	getopt()	prealloc()	sysconf()
execl()	getpass()	read()	tcgetpgrp()
execle()	getpgrp()	readlink()	tcsetpgrp()
execlp()	getpgrp2()	rmdir()	truncate()
execv()	getpid()	sbrk()	ttyname()
execve()	getppid()	setgid()	ttyslot()
execvp()	getuid()	setgroups()	unlink()
_exit()	getusershell()	sethostname()	vfork()
fchown()	hidecdf()	setpgid()	write()

SEE ALSO

access(2), chown(2), confstr(3C), exit(2), fcntl(2), kill(2), lseek(2), open(2), pathconf(2), sysconf(2), limits(5), stdsyms(5), termio(7).

AUTHOR

`unistd` was developed by HP.

STANDARDS CONFORMANCE

`<unistd.h>`: AES, XPG2, XPG3, XPG4, FIPS 151-2, POSIX.1, POSIX.2

NAME

values - machine-dependent values

SYNOPSIS

#include <values.h>

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

BITS(*type*)	The number of bits in a specified type (e.g., int).
HIBITS	The value of a short integer with only the high-order bit set (in most implementations, 0x8000).
HIBITL	The value of a long integer with only the high-order bit set (in most implementations, 0x80000000).
HIBITI	The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).
MAXSHORT	The maximum value of a signed short integer (in most implementations, 0x7FFF ≡ 32767).
MAXLONG	The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF ≡ 2147483647).
MAXINT	The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).
MAXFLOAT,	LN_MAXFLOAT The maximum value of a single-precision floating-point number, and its natural logarithm.
MAXDOUBLE,	LN_MAXDOUBLE The maximum value of a double-precision floating-point number, and its natural logarithm.
MINFLOAT,	LN_MINFLOAT The minimum positive value of a single-precision floating-point number, and its natural logarithm.
MINDOUBLE,	LN_MINDOUBLE The minimum positive value of a double-precision floating-point number, and its natural logarithm.
FSIGNIF	The number of significant bits in the mantissa of a single-precision floating-point number.
DSIGNIF	The number of significant bits in the mantissa of a double-precision floating-point number.

FILES

/usr/include/values.h

SEE ALSO

intro(3), math(5).

STANDARDS CONFORMANCE

`<values.h>`: XPG2

NAME

varargs - handle variable argument list

SYNOPSIS

```
#include <varargs.h>

va_alist

va_dcl

void va_start(pvar)
va_list pvar;

type va_arg(pvar, type)
va_list pvar;

void va_end(pvar)
va_list pvar;
```

DESCRIPTION

This set of macros enables programmers to write portable procedures that accept variable argument lists. Routines that have variable argument lists (such as **printf()**) but do not use **varargs** are inherently nonportable, because different machines use different argument-passing conventions (see *printf*(3S)).

va_alist is used as the parameter list in a function header.

va_dcl is a declaration for **va_alist**. No semicolon should follow **va_dcl**.

va_list is a type defined for the variable used to traverse the list.

va_start is called to initialize **pvar** to the beginning of the list.

va_arg returns the next argument in the list pointed to by **pvar**. *type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, because it cannot be determined at runtime.

va_end is used to clean up.

Multiple traversals, each bracketed by **va_start** ... **va_end**, are possible.

EXAMPLE

The following example shows a possible implementation of **execl()** (see *exec*(2)):

```
#include <varargs.h>
#define MAXARGS 100

    /*  execl is called by
        execl(file, arg1, arg2, ..., (char *)0);
    */
    execl(va_alist)
    va_dcl
    {
          va_list ap;
          char *file;
          char *args[MAXARGS];
          int argno = 0;

          va_start(ap);
          file = va_arg(ap, char *);
          while ((args[argno++] = va_arg(ap, char *)) != (char *)0);

          va_end(ap);
          return execv(file, args);
    }
```

The next example illustrates how a function that receives variable arguments can pass these arguments down to other functions. To accomplish this, the first routine (**log_errors()** in this example) which receives the variable argument list must pass the address pointer resulting from a call to **va_start()** on to any subsequent calls that need to access this same variable argument list. All routines that receive this

address pointer (`v_print_log()` in this example) need only to use `va_arg()` to access the original variable argument list just as if they were the original routine to be passed the variable arguments.

In this example, one can imagine that there are a series of other routines (such as a `log_warning()` and `log_message()`) that also call the `v_print_log()` function.

```
#include <stdio.h>
#include <varargs.h>
#include <unistd.h>

int error_count;

/* VARARGS4 -- for lint */
int
log_errors(log_fp, func_name, err_num, msg_fmt, va_alist)
FILE *log_fp;
char *func_name;
int err_num;
char *msg_fmt;
va_dcl
{
    va_list ap;

    /* Print error header information */
    (void) fprintf(log_fp, "0RROR in process %d0, getpid());
    (void) fprintf(log_fp, "  function
    switch(err_num)
    {
        case ILLEGAL_OPTION:
            (void) fprintf(log_fp, "illegal option0);
            break;
        case CANNOT_PARSE:
            (void) fprintf(log_fp, "cannot parse input file0);
            break;
        ...
    }

    /*
    * Get pointer to first variable argument so that we can
    * pass it on to v_print_log().  We do this so that
    * v_print_log() can access the variable arguments passed
    * to this routine.
    */
    va_start(ap);

    v_print_log(log_fp, msg_fmt, ap);

    va_end(ap);
}

/* VARARGS2 -- for lint */
int
v_print_log(log_fp, fmt, ap)
FILE *log_fp;
char *fmt;
va_list ap;
{

    /*
     * If "%Y" is the first two characters in the format string,
     * a second file pointer has been passed in to print general
     * message information to.  The rest of the format string is
     * a standard  printf(3S) format string.
     */
```

```
        if ((*fmt == '%') && (*(fmt + 1) == 'Y'))
        {
            FILE *other_fp;

            fmt += 2;

            other_fp = (FILE *) va_arg(ap, char *);
            if (other_fp != (FILE *) NULL)
            {
                /*
                 * Print general message information to additional stream.
                 */
                (void) vfprintf(other_fp, fmt, ap);
                (void) fflush(other_fp);
            }
        }

        /*
         * Now print it to the log file.
         */
        (void) vfprintf(log_fp, fmt, ap);
    }
```

SEE ALSO

exec(2), vprintf(3S).

BUGS

It is up to the calling routine to specify how many arguments there are, because it is not always possible to determine this from the stack frame. For example, `execl()` is passed a zero pointer to signal the end of the list. `printf()` can determine how many arguments are present by the format.

It is non-portable to specify a second argument of `char`, `short`, or `float` to `va_arg`, because arguments seen by the called function are not `char`, `short`, or `float`. C converts `char` and `short` arguments to `int`, and converts `float` arguments to `double`, before passing them to a function.

STANDARDS CONFORMANCE

<varargs.h>: AES, XPG2, XPG3, XPG4
va_alist: AES, SVID2, XPG2, XPG3, XPG4

va_arg: SVID2, XPG2, XPG3, XPG4
va_dcl: SVID2, XPG2, XPG3, XPG4

va_end: SVID2, XPG2, XPG3, XPG4
va_list: SVID2, XPG2, XPG3, XPG4

va_start: SVID2, XPG2, XPG3, XPG4

NAME

X_Open - Pointer manual entry for X/Open XPG3 Conformance Statements

DESCRIPTION

This entry is a pointer entry for accessing X/OPEN XPG3 Conformance Statements for HP9000 Series 300/400 and Series 700/800 HP-UX systems. To access the conformance statement for Series 300/400 systems, use the command:

man X_Open_300

For Series 700 or 800 systems, use the command:

man X_Open_800

Compliance Exception

The **encrypt()** and **setkey()** functions shipped with standard HP-UX Release 8.0 are not fully X/Open compliant. This characteristic is not of concern to most users, but may be a problem for programmers who are concerned about 100% X/Open compliance and/or who have applications that use **encrypt()** and expect X/Open-compliant behavior.

On HP-UX systems that are suitable for export from the USA, **encrypt()** has been modified so that it cannot decrypt previously encrypted data. When a fully functional **encrypt()** cannot be provided, X/Open requires that **encrypt()** and **setkey()** return ENOSYS instead. For systems that are not allowed to have fully functional **encrypt()** and **setkey()** but still require X/Open compliance, a patch is available that replaces the partially functional **encrypt()** and **setkey()** with X/Open compliant versions that do nothing and return the required ENOSYS. The patch does not alter the normal behavior of **crypt()**.

To obtain the patch, contact your local HP Sales and Support office and request the following patch:

Series 300/400:	Patch Number PHCO_0419
Series 700:	Patch Number PHCO_0455
Series 800:	Patch Number PHCO_0456

U.S. Domestic customers who need a fully functional (and also X/Open compliant) version of these routines should contact their local HP Sales and Support Office for more information.

FILES

/usr/man/man5.Z/X_Open_300.5	Series 300 Conformance Statement (preformatted and compressed)
/usr/man/man5.Z/X_Open_800.5	Series 800 Conformance Statement (preformatted and compressed)

Section 7:
Device (Special) Files

NAME

intro - introduction to special files

DESCRIPTION

This section describes various special files that refer to specific HP peripherals and device drivers. The names of the entries are generally derived from the type of device being described (disk, plotter, etc.), not the names of the special files themselves. Characteristics of both the hardware device and the corresponding HP-UX device driver are discussed where applicable.

The devices are divided into two categories, **unblocked** and **blocked.** An unblocked device is also called a **raw** or character-mode device. An unblocked device such as a line printer uses a character special file.

Blocked devices, as the name implies, transfer data in blocks by means of the system's normal buffering mechanism. Block devices use block special files.

For specific details about the default special files shipped with your system, consult the system administrator manuals for your system.

A name becomes associated with a specific device when the special file is created for that device by using the **mkdev** script or **mknod** command (see *mkdev*(1M) and *mknod*(1M)). When creating special files, it is recommended that the following naming convention be followed. For disk and tape, it is identical with that used on other UNIX systems, and is independent of the hardware.

The following format is for tape device file names:

/dev/rmt/(c#d)#[hml][c][n][b]

where **c#d** indicates the controller number (which is optionally specified by the system administrator), **#** is the device number, **hml** indicates the density (**h** (high) for 6250 bpi, **m** (medium) for 1600 bpi, and **l** (low) for 800 bpi), **c** indicates data compression, **n** indicates no rewind on close and **b** indicates Berkeley style device, e.g., **/dev/rmt/2mnb**.

The following format is for QIC tape device file names:

/dev/rmt/(c#d)#qic[525 | 150 | 120][n][b]

where **c#d** indicates the controller number (which is optionally specified by the system administrator), **#** is the device number, **qic525 | 150 | 120** indicates the QIC format (**qic** (without a format number, i.e. default format) specifies the best density format for the drive and currently loaded medium, **qic525** for QIC-525/320 format, **qic150** for QIC-150 format, and **qic120** for QIC-120 format), **n** indicates no rewind on close and **b** indicates Berkeley style device, e.g., **/dev/rmt/2qic525nb**.

The following format is for hard disk device file names:

/dev/[r]dsk/(r)(c#d)#s#

where **r** indicates a raw interface to the disk, the second **r** indicates that this disk is on a remote system, the **c#d** indicates the controller number (which is optionally specified by the system administrator), and **#s#** indicates the drive and section numbers, respectively.

WARNINGS

Several other naming conventions have been used in the past for given devices. Using *ln*(1) to create a link between the old name and the new standard name is useful as a temporary expedient until all programs using the old naming convention have been converted.

In general, device drivers are not portable across systems. However, every effort has been made to make their behavior portable. Due to variation in hardware, this is not always possible. Programs that use these drivers directly are at higher-than-average risk of not being portable.

SEE ALSO

hier(5).

The introduction to this manual.

The system administrator manual for your system.

NAME

AF_CCITT - CCITT address family

SYNOPSIS

#include <sys/types.h>
#include <x25/ccittproto.h>

DESCRIPTION

AF_CCITT identifies protocols within the address-family CCITT, such as X.25.

Addressing

The include file **<x25/x25addrstr.h>** contains the declaration of the **x25addrstr struct**. Pointers to **x25addrstr** are used in various system calls.

There are six fields of interest within this structure. The first field is *x25_family*, which must be set to **AF_CCITT**. The second field is *hostlen* which contains the length of the X.121 address in digits. The third field is *pidlen* which contains the length of the *x25pid* field in bytes. The fourth field is ®which specifies a protocol-ID. A protocol-ID is a loosely-defined convention designating a number of bits in the Call-user data field of an INCOMING CALL packet as the protocol-ID. The protocol-ID is specified in 1 to 8 bytes. The fifth field is *x25_host*[], which is the X.121 address of the destination for a connection request, or the address to which the socket is bound. The X.121 address is a string of 1 to 15 digits, including subaddesses if subaddesses are supported on the network provider. The sixth field is *x25ifname*[], the name of the local X.25 interface through which a connection will be established, or to which the socket is to be bound. The X.25 interface name is a string of 1 to 12 alphanumeric characters, terminated by the null character.

Socket Buffer Size

The maximum buffer size for a stream socket in the AF_CCITT family is 65 535 bytes. The default buffer size is 4096 bytes. The send and receive buffer sizes for AF_CCITT stream sockets can be altered by using the **SO_SNDBUF** and **SO_RCVBUF** options of the **setsockopt()** system call. See *getsockopt*(2) for details.

AUTHOR

af_ccitt was developed by the University of California, Berkeley.

SEE ALSO

accept(2), bind(2), connect(2), getpeername(2), getsockname(2), getsockopt(2), socketx25(7), tcp(7P).

NAME

arp - Address Resolution Protocol

DESCRIPTION

ARP is a protocol used to dynamically map between DARPA Internet and hardware station addresses. It is used by all LAN drivers.

ARP caches Internet-to-hardware station address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message that requires the mapping, and broadcasts a message on the associated network requesting the address mapping if the **ether** encapsulation method has been enabled for the interface. If a response is provided, the new mapping is cached and any pending message is transmitted. ARP queues at most one packet while waiting for a mapping request to be responded to; only the most recently "transmitted" packet is kept.

To facilitate communications with systems that do not use ARP, ioctls are provided to enter and delete entries in the Internet-to-hardware station address tables. Usage:

```
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/if_ether.h>
struct arpreq arpreq;

ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```

Each ioctl takes the same structure as an argument. **SIOCSARP** sets an ARP entry, **SIOCGARP** gets an ARP entry, and **SIOCDARP** deletes an ARP entry. These ioctls can be applied to any socket descriptor *s*, but only by the super-user. The **arpreq** structure contains:

```
/*
 * ARP ioctl request
 */
struct arpreq {
        struct sockaddr arp_pa;         /* protocol address */
        struct sockaddr arp_ha;         /* hardware address */
        u_char rif[18];                 /* 802.5 source routing information */
        int     arp_flags;              /* flags */
};
/*  arp_flags field values */
#define ATF_COM         0x02        /* ARP on ether */
#define ATF_PERM        0x04        /* permanent entry */
#define ATF_PUBL        0x08        /* publish entry */
#define ATF_USETRAILERS 0x10        /* trailer packets */
#define ATF_ETHERXT     0x20        /* PROBE on ether */
#define ATF_IEEE8023    0x40        /* PROBE on ieee */
#define ATF_SNAPFDDI    0x200       /* SNAP - FDDI */
#define ATF_SNAP8025    0x400       /* SNAP - 8025 */
#define ATF_IEEE8025    0x800       /* IEEE - 8025 */
```

The address family for the *arp_pa* sockaddr must be **AF_INET**; for the *arp_ha* sockaddr it must be **AF_UNSPEC**. The only flag bits that can be written are **ATF_PERM**, **ATF_PUBL**, and **ATF_USETRAILERS**. **ATF_PERM** causes the entry to be permanent. **ATF_PUBL** specifies that the ARP code should respond to ARP requests for the indicated host coming from other machines. This allows a host to act as an *ARP server*, which may be useful in convincing an ARP-only machine to talk to a non-ARP machine.

ARP is also used to negotiate the use of trailer IP encapsulations; trailers are an alternate encapsulation used to allow efficient packet alignment for large packets despite variable-sized headers. Hosts that want to receive trailer encapsulations indicate so by sending gratuitous ARP translation replies along with replies to IP requests; they are also sent in reply to IP translation replies. The negotiation is thus fully symmetrical, in that either or both hosts can request trailers. The **ATF_USETRAILERS** flag is used to record the receipt of such a reply, and enables the transmission of trailer packets to that host.

ARP watches passively for hosts impersonating the local host (i.e., a host that responds to an ARP mapping request for the local host's address).

AUTHOR

ARP was developed by the University of California, Berkeley.

WARNINGS

HP 9000 systems can receive trailer packets but do not send them. Setting the `trailers` flag has no effect.

To enable the 'ether' encapsulation method, use the `lanconfig` command (see *lanconfig*(1M)).

DIAGNOSTICS

`duplicate IP address!! sent from ethernet address: %x:%x:%x:%x:%x:%x.`
This message printed on the console screen means that ARP has discovered another host on the local network that responds to mapping requests for its own Internet address.

SEE ALSO

ifconfig(1M), inet(3N), lanconfig(1M), lan(7), arp(1M).
An Ethernet Address Resolution Protocol, RFC826, Dave Plummer, Network Information Center, SRI.
Trailer Encapsulations, RFC893, S.J. Leffler and M.J. Karels, Network Information Center, SRI.

NAME

autochanger - optical autochanger driver

SYNOPSIS

```
#include <sys/autoch.h>

int ioctl(int fildes, int request, ... /* arg */);
```

DESCRIPTION

An autochanger is a mass storage device with a very large capacity. It consists of one or two drives, media, and a mechanical changer. The mechanical changer allows multiple pieces of media to be shared among the drives. Each piece of media has two surfaces, but only one side can be accessed in a drive. The media must be inverted in order to access data on the reverse side.

The autochanger driver orchestrates swapping of media in and out of the drives such that it appears to the user as if each piece of media has its own drive.

Two daemons named **xportd** and **spinupd** are automatically started when the autochanger is first accessed. Each autochanger has its own unique pair of daemons.

Series 300/400 and Series 700 autochanger media device files are character special files with a major number of 55 or block special files with a major number of 10.

Series 800 Major Numbers

Series 800 autochanger media device files are character special files with a major number of 19 or block special files with a major number of 12.

Series 300/400 Minor Numbers

The minor number for autochanger media takes the form **0x***ScAFSr* where *ScAFSr* is constructed as follows:

Sc Select code of the mechanical changer
A SCSI address of the mechanical changer
F Reserved. The only valid value is zero.
Sr Surface for the specified media, encoded as follows:

Surface	Value of *Sr*
surface1a	01
surface1b	02
surface2a	03
surface2b	04
	.
	.
	.
surface32a	3F
surface32b	40

The character device file with an *Sr* value of **00** is used for **ioctl()** requests (see *ioctl*(2)). This device must be opened before any request can be performed (see EXAMPLES).

Series 700 Minor Numbers

The minor number for autochanger media takes the form **0x***VEAFBSr* where *VEAFBSr* is a 24-bit value constructed as follows:

V VSC slot number (bits 23-21).
E Function exists (bit 20).
A Adapter slot number (bits 16-19).
F Function number (bits 12-15).
B SCSI bus address of the autochanger (bits 09-11). Because the surface numbers do not end on an even boundry (9 bits specified) the autochanger SCSI bus address is shifted one bit to the left (see example below).
Sr Surface for the specified media (bits 00-08 — encoded same as Series 300 above).

Examples of surface devices for Series 700 systems connected to the internal SCSI bus with the autochanger SCSI ID at 3:

```
mknod /dev/rac/1a 55 0x201601 # Cartridge
mknod /dev/rac/1b 55 0x201602 # Cartridge
mknod /dev/rac/2a 55 0x201603 # Cartridge
mknod /dev/rac/2b 55 0x201604 # Cartridge
```

Series 800 Minor Numbers

The minor number for autochanger media takes the form `0x`*LuPFSr* where *LuPFSr* is constructed as follows:

Lu Logical unit number of the mechanical changer.

P Partition (section). See *disk*(7) for definition.

F Reserved. The only valid value is zero.

Sr Surface for the specified media (bits 00-08 — encoded same as Series 300 above).

The character device file with an *Sr* value of `00` is used for `ioctl()` requests (see *ioctl*(2)). This device must be opened before any request can be performed. (see EXAMPLES).

ioctl commands

The following `ioctl()` commands (see the header file <`sys/autoch.h`>) are valid:

`ACIOC_WRITE_Q_CONST`
This request alters the constants that control the queue.

`ACIOC_READ_Q_CONST`
This request reads the constants that control the queue.

A pointer to a structure `queue_const` is passed as the *arg* argument to `ioctl()` for `ACIOC_READ_Q_CONST` and `ACIOC_WRITE_Q_CONST` requests, as defined in header file <`sys/autoch.h`>:

```
struct queue_const {
        int wait_time;
        int hog_time;
};
```

where:

`wait_time` (in seconds) controls how long the driver waits for another request for the current surface before swapping. When the device is first opened, the default is 1 second.

`hog_time` How many seconds a surface can be held in a drive while requests for another surface are pending. When the device is first opened, the default is 20 seconds.

The value of `wait_time` cannot exceed that of `hog_time`.

`ACIOC_READ_Q_STATS`
This `ioctl()` reads the vital statistics of the queue.

A pointer to a structure `queue_stats` is passed as the *arg* argument to `ioctl()` for the `ACIOC_READ_Q_STATS` request, as defined in header file <`sys/autoch.h`>:

```
struct queue_stats {
        long size;
        long mix;
};
```

`ACIOC_INITIALIZE_ELEMENT_STATUS`
This request instructs the autochanger to reset its internal media map, and causes the mechanical changer to look for media in all of the slots. It fails if any surface is open.

RETURN VALUE

Upon successful completion, `ioctl()` returns a value of zero. If an error occurs, a value of `-1` is returned and the global variable `errno` is set to indicate the error.

OTHER SUPPORTED IOCTLS

These ioctls can be used with surface devices and are *not* to be used on the `/dev/rac/ioctl` device:

```
DIOC_EXCLUSIVE
DIOC_CAPACITY
DIOC_DESCRIBE
SIOC_INQUIRY
SIOC_XSENSE
```

ERRORS

In addition to those errors defined in *open*(2), *close*(2), *read*(2), *write*(2), and *ioctl*(2), a driver request can fail if any of the following conditions are encountered:

[EIO]	`ioctl()` encountered a hardware problem with the mechanical changer or drive.
[ENXIO]	The device file is incorrect, or a surface was open while `ioctl()` attempted to execute the `ACIOC_INITIALIZE_ELEMENT_STATUS` request.

EXAMPLES

The following code fragment reads the queue constants for an autochanger device accessed through special file `/dev/rac/ioctl` (the special device file with *Sr* = 00):

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/autoch.h>

int fildes;
struct queue_stats ac_queue_stats;

/* open the special device file for ioctl */
if ((fildes = open("/dev/rac/ioctl", O_RDWR)) < 0)
    perror(...);
/* do the ioctl */
if (ioctl(fildes, ACIOC_READ_Q_CONSTS, &ac_queue_stats) < 0)
    perror(...);
/* print the results */
(void) printf("Queue size is %d\n", ac_queue_stats.size);
(void) printf("Queue mix is %d\n", ac_queue_stats.mix);
```

DEPENDENCIES

This driver supports only the C1700 and C1702 Optical Autochangers.

AUTHOR

The `autochanger` driver was developed by HP.

FILES

`/dev/ac/*`	block special files
`/dev/rac/*`	character special files

SEE ALSO

insf(1M), mkdev(1M), mknod(1M), ioctl(2).
HP-UX System Administrator Manuals.

NAME

blmode - terminal block mode interface

DESCRIPTION

This terminal interface adds functionality to the current *termio*(7) functionality to allow for efficient emulation of MPE terminal driver functionality. Most importantly, it adds the necessary functionality to support block mode transfers with HP terminals. The block mode interface only affects input processing and does not affect write requests. Write requests are always processed as described in *termio*(7). In character mode the terminal sends each character to the system as it is typed. However, in block mode data is buffered and possibly edited locally in the terminal memory as it is typed, then sent as a block of data when the **Enter** key is pressed on the terminal. During block mode data transmissions, the incoming data is not echoed and no special character processing is performed, other than recognizing a data block terminator character. For subsequent character mode transmissions, the existing termio state continues to determine echo and character processing.

There are two parts of the block mode protocol. The first part is the block mode handshake, which works as follows:

- At the beginning of a read, a *trigger* character is sent to the terminal to notify it that the system is requesting a block of data. (The *trigger* character, if defined, is sent at the beginning of all reads, whether character or block. The *trigger* character must be defined for block mode reads.)
- After receiving the *trigger* character, the terminal waits until the user has typed data into the terminal's memory and pressed the terminal **Enter** key. The terminal then sends an *alert* character to the system to notify it that the terminal has a block of data to send.
- The system may then send user-definable cursor positioning or other data sequences to the terminal. When that is done, the system sends another *trigger* character to the terminal, repeating the cycle.

The second part of the block mode protocol is the block mode transmission. During this transmission of data, the incoming data is not echoed and no special character processing is performed, other than recognizing the data block termination character. It is possible to bypass the block mode handshake and have the block mode transmission occur after the first *trigger* character is sent.

To prevent data loss, XON/XOFF flow control should be used between the system and the terminal. The IXOFF bit should be set and the terminal strapped appropriately. If flow control is not used, it is possible for incoming data to overflow and be lost. (Note: some older terminals do not deal correctly with this flow control.)

It is possible to intermix both character mode and block mode data transmissions. If block mode transmissions are enabled, all transfers are handled as block mode transfers. When block mode transmissions are not enabled, character mode transmissions are processed as described in *termio*(7). If block mode transmissions are not enabled, but an *alert* character is received anywhere in the input data, the transmission mode is switched to block mode automatically for a single transmission.

Read requests that receive data from block mode transmissions will not be returned until the transmission is complete; i.e., the terminal has transmitted all characters. If the read is satisfied by byte count or if a data transmission error occurs, any subsequent data will be discarded. The read waits until completion of the data transmission before returning.

The data block terminator character is included in the data returned to the user, and is included in the byte count. If the number of bytes transferred by the terminal in a block mode transfer exceeds the number of bytes requested by the user, the read returns the requested number of bytes, and the remaining bytes are discarded. The user can determine if data was discarded by checking the last character of the returned data. If the last character is not the terminator character, more data was received than was requested, and data was discarded.

If desired, the application program can provide its own handshake mechanism in response to the *alert* character by selecting the OWNTERM mode. With this mode selected, the driver completes a read request when the *alert* character is received. The second *trigger* is sent by the driver when the application issues the next read.

Several special characters (both input and output) are used with block mode. These characters and the normal values used for block mode are described below. The initial value for these characters is 0377, which

causes them to be disabled.

CBTRIG1C (DC1) is the initial *trigger* character sent to the terminal at the beginning of a read request.

CBTRIG2C (DC1) is the secondary *trigger* character sent to the terminal after the *alert* character has been received.

CBALERTC (DC2) is the *alert* character sent by the terminal in response to the first *trigger* character. It signifies that the terminal is ready to send the data block. The *alert* character can be escaped by preceding it with a backslash (\).

CBTERMC (RS) is sent by the terminal after the block mode transfer is complete. It signifies the end of the data block to the computer.

The two *ioctl*(2) requests that apply to block mode use the **blmodeio** structure, which defined in <**blmodeio.h**>, and includes the following members:

```
unsigned long   cb_flags;     /* Modes */
unsigned char   cb_trig1c;    /* First trigger */
unsigned char   cb_trig2c;    /* Second trigger */
unsigned char   cb_alertc;    /* Alert character */
unsigned char   cb_termc;     /* Terminating char */
unsigned char   cb_replen;    /* cb_reply length */
char            cb_reply[ ];  /* optional reply */
```

The *cb_flags* member controls the basic block mode protocol:

CB_BMTRANS	0000001	Enable mandatory block mode transmission.
CB_OWNTERM	0000002	Enable user control of handshake.

The CB_BMTRANS bit is only effective when the ICANON flag in *termio*(7) is set. If ICANON is clear, all transfers are done in raw mode, regardless of the CB_BMTRANS bit. If CB_BMTRANS is not set, input processing is performed as described in *termio*(7). During this time, if the *alert* character is defined and is detected anywhere in the input stream, the input buffer is flushed and block-mode handshake is invoked. The system then sends the *cb_trig2c* character to the terminal, and a block mode transfer follows. The *alert* character can be escaped by preceding it with a backslash (\).

If CB_BMTRANS is set, then all transmissions are processed as block mode transmissions. Block mode handshake is not required and data read is processed as block mode transfer data. Block mode handshake can still be invoked by receipt of an *alert* character as the first character received. Reads issued while the CB_BMTRANS bit is set cause any existing input buffer data to be flushed.

If CB_OWNTERM is set, reads are terminated upon receipt of a non-escaped *alert* character. No input buffer flushing is performed and the *alert* character is returned in the data read. This allows application code to perform its own block-mode handshaking. If the bit is clear, an *alert* character causes normal block mode handshaking to be used.

The initial **cb_flags** value is all-bits-cleared.

The **cb_trig1c** character is the initial *trigger* character sent to the terminal at the beginning of a read request. The initial value is undefined (0377); i.e., no *trigger* character is sent.

The **cb_trig2c** character is the secondary *trigger* character sent to the terminal after the *alert* character has been received. The initial value is undefined (0377).

The **cb_alertc** character is the *alert* character sent by the terminal in response to the first *trigger* character sent by the computer. It signifies that the terminal is ready to transmit data. The initial value is undefined (0377).

The **cb_termc** character is sent by the terminal after the block mode transfer has completed. It signifies the end of the data block to the computer. The initial value is undefined (0377).

The **cb_replen** member specifies the length in bytes of the **cb_reply** array. The maximum length of the **cb_reply** array is **NBREPLY** bytes. If set to zero, the *cb_reply* string is not used. It is initially set to zero.

The *cb_reply* array contains a string to be sent out after receipt of the *alert* character but before the second *trigger* character is sent by the computer. Any character can be included in the reply string. The number of

characters sent is specified by **cb_replen.** The maximum length of the **cb_reply** array is **NBREPLY** bytes. The initial value of all characters in the **cb_reply** array is null.

On systems that support process group control, *ioctl* requests are restricted from use by background processes, unless otherwise noted for a specific request. An attempt to issue an *ioctl* request from a background process causes the process to block and may cause a SIGTTOU signal to be sent to the process group.

The primary *ioctl*(2) calls have the form:

int ioctl(int fildes, int request, struct blmodeio *arg);

Requests using this form include:

CBGETA Get the parameters associated with the block mode interface and store them in the *blmodeio* structure referenced by *arg*. This request is allowed from a background process. However, the information may be subsequently changed by a foreground process.

CBSETA Set the parameters associated with the block mode interface from the *blmodeio* structure referenced by *arg*. The change is immediate.

RETURN VALUE

Refer to *read*(2), *write*(2), and *ioctl*(2).

ERRORS

If an error value is returned during a read, it is possible for the user's buffer to be altered. In this case, the data in the user's buffer should be ignored because it is incomplete.

The global variable *errno* will be set to indicate the following error, in addition to those errors described on *read*(2), *write*(2), and *ioctl*(2):

[EIO] A read error occurred during the transmission of the block mode data block.

WARNINGS

The EIO error that is returned for read errors can be caused by many events. The read returns EIO for transmission, framing, parity, break, and overrun errors, or if the internal timer expires. The internal timer starts when the second *trigger* character is sent by the computer, and ends when the terminating character is received by the computer. The length of this timer is determined by the number of bytes requested in the read and the current baud rate, plus an additional ten seconds.

AUTHOR

The *blmode* driver was developed by HP.

SEE ALSO

termio(7).

NAME

cent - Centronics-compatible interface

REMARKS

This manual entry applies only to peripherals connected to the `cent` interfaces found on Series 300, 400, and 700 systems.

DESCRIPTION

`cent` is a simple, widely used communication protocol most commonly associated with printers, plotters and scanners. It is an eight-bit parallel data interface with additional control signals from the host computer, and status signals from the peripheral.

The model for the `cent` interface drivers is simple in that no character processing is done; i.e., no interpretation is applied to the data being transferred between computer and peripheral. Therefore, all bytes sent to or received from a device are handled without alteration. In other words, the `cent` interface drivers always operate in **raw mode** which means that any desired data interpretation must be performed by a user program (such as the lp spooler in conjunction with an appropriate model file).

DEPENDENCIES

Series 300/400

A wide variety of control functions are supported. See Device I/O Library (*gpio**(3I), *hpib**(3I), and *io**(3I)), entries in Section 3 of this manual.

Series 700

The Series 700 `cent` driver supports six different handshake modes for data transfer. The last four bits of the minor number of the device special file specify which mode is used. The format of the device minor number is:

`0x`*SEF00A*

where each letter represents a single hexadecimal digit as follows:

`0x` Preamble to a hexadecimal number.

S Specifies the system bus module number (2 for core I/O).

E Specifies the EISA slot number: (always 0 for core I/O).

F Specifies the function number. Always 6 for the Core I/O parallel interface.

`00` Always 0.

A Specifies the handshake mode. The handshake modes are:

- mode 1 Automatic handshaking using both nACK and BUSY. Minor number format: `0x`*SEF*`001`.
- mode 2 Automatic handshaking using only BUSY. Minor number format: `0x`*SEF*`002`.
- mode 3 Bidirectional read/write used for ScanJet. Minor number format: `0x`*SEF*`003`.
- mode 4 Stream mode. Data is essentially transmitted to the peripheral without any handshaking protocol. Minor number format: `0x`*SEF*`004`.
- mode 5 Pulsed mode using both ACK and BUSY for automatic handshaking. Similar to mode 1 except that the data strobe line, nSTROBE is pulsed for a fixed amount of time by the sender, then released. Minor number format: `0x`*SEF*`005`.
- mode 6 Pulsed mode, using only BUSY for automatic handshaking. Similar to mode 1 except that the data strobe line, nSTROBE is pulsed for a fixed amount of time by the sender, then released. Minor number format: `0x`*SEF*`005`.

Modes 1 and 2 support most printers belonging to the *Jet series (LaserJet, DeskJet, QuietJet, etc.).

AUTHOR

`cent` was developed by HP.

SEE ALSO

lp(1), ioctl(2), intro(7), lp(7).

System Adminstrator manuals included with your system.

NAME

console - system console interface

DESCRIPTION

/dev/console is a generic name given to the system console. It is usually linked to a particular machine-dependent special file. It provides a basic I/O interface to the system console through the *termio* interface.

SEE ALSO

termio(7).

STANDARDS CONFORMANCE

console: SVID2, XPG2

NAME

ct - Command-Set 80 (CS/80) cartridge tape access

DESCRIPTION

This entry describes the actions of the general HP-UX Command-Set 1980 (CS/80) cartridge tape drivers when referring to a CS/80 cartridge tape as either a block- or character-special (raw) device.

Cartridge tapes are designed to work optimally as "streaming" devices, and are not designed to start and stop frequently. Technically, they are "random access" devices such as disks, but such access is both less efficient and more stressful than streaming mode. Thus it is possible to use a cartridge tape as a file system, or in general access it randomly, but such use will more rapidly wear either or both the tape drive and the media.

Cartridge tape units in either CS/80 disk drives or in stand-alone devices can be accessed as blocked or raw devices.

Block special files access cartridge tapes via the system's normal buffering mechanism. Buffering is done in such a way that concurrent access through multiple opens or a mount of the same physical device do not get out of phase. Block special files may be read and written without regard to physical cartridge tape records. Each I/O operation results in one or more logical block transactions. In general, this mode is not recommended as it stresses the hardware.

There is also a *raw* interface via a character special file which provides for direct transmission between the cartridge tape and the user's read or write buffer. A single read or write operation results in exactly one transaction. Therefore raw I/O is considerably more efficient when many bytes are transmitted in a single operation because blocked cartridge tape access requires potentially several transactions and does not transmit directly to user space.

In raw I/O, there may be implementation dependent restrictions on the alignment of the user buffer in memory and its maximum size. Also, each transfer must occur on a record boundary, and must read a whole number of records. The record size is a hardware-dependent value.

Selecting the proper buffer size when accessing a cartridge tape device through the raw interface is critical to the performance of the cartridge tape device and other devices connected on the same HPIB. A large buffer in certain situations can increase performance but has the potential to block other devices on the HPIB until all the data for a request has been transferred. On the other hand when a small buffer is used and the application is unable to keep the cartridge tape device streaming, performance and the wear and tear of the device suffer because of tape repositioning. The optimal solution is to keep the tape streaming while using a small buffer. To select the proper buffer size, consider two factors: the cartridge tape device being accessed and the application which is accessing the cartridge tape device.

Some cartridge tape units (see DEPENDENCIES) support a feature called immediate report mode. During writing, this mode enables the drive to complete a write transaction with the host before the data has actually been written to the tape from the drive's buffer. This allows the host to start gathering data for the next write request while the data for the previous request is still in the process of being written. During reading, this mode enables the drive to read ahead after completing a host read request. This allows the drive to gather data for future read requests while the host is still processing data from the previous read request. When data is requested or supplied at a sufficient rate, immediate report mode allows the drive to stream the tape continuously across multiple read/write requests, as opposed to having to reposition the tape between each read/write request. Repositioning adds to the wear and tear of the cartridge tape device and decreases the performance. Some cartridge tape devices (see DEPENDENCIES) do not support immediate report mode and as such cannot stream across multiple requests.

If the cartridge tape device being accessed supports immediate report mode and the application can maintain a data rate that allows the cartridge tape device to stream multiple requests, a small buffer (1 Kbyte to 12 Kbytes) is suggested so that the HP-IB is not blocked for a significant amount of time. For cartridge tape devices that do not support immediate report mode or applications that cannot maintain a data rate that allows the cartridge tape device to stream multiple requests, a large buffer (64 Kbytes) is suggested so that the number of tape repositions is reduced.

Each raw access is independent of other raw accesses and of block accesses to the same physical device. Thus, transfers are not guaranteed to occur in any particular order. Having multiple programs access the cartridge tape is, in effect, random access, and is subject to the warnings above.

In raw I/O, each operation is completed to the device before the call returns. For block-mode writes, the data may be cached until it is convenient for the system to write it. In addition, block-mode reads potentially do a one (or more) block read-ahead. The interaction of block-mode and raw access to the same cartridge tape is not specified, and in general is unpredictable. Because block-mode writes can be delayed, it is possible for a program to generate requests much more rapidly than the drive can actually process them. Flushing a large number of requests could take several minutes, and during that time the system will not have use of the buffers taken by these requests, and thus will suffer a possibly severe performance degradation. If the tape is integral with the system disk, very little disk activity may be possible until the buffers are flushed.

Cartridge tape device file names are in the following format:

```
/dev/[r]ct/[r]c#[d#][s#]
```

where the first `r` indicates a raw interface to the cartridge tape, the second `r` is reserved to indicate that this cartridge tape is on a remote system, the `c#` indicates the controller number, the `d#` optionally indicates the drive, and the `s#` optionally indicates a section number. The assignment of controller, drive, and section numbers is described in the system administrator's manual for your system.

WARNINGS

Like disks, the cartridge tape units in CS/80 disk drives can be accessed as blocked or raw devices. However, using a cartridge tape as a file system severely limits the life expectancy of the tape drive. Tapes should be used only for system back-up and other needs where data must be stored on tape for transport or other purposes.

`ct` does not support access of DDS and QIC cartridge tape devices.

DEPENDENCIES

HP 7941CT/HP 9144A/HP 35401

These cartridge tape devices support the immediate report mode.

HP 7942/HP 7946

These cartridge tape devices support the immediate report mode. The use of a small buffer size is not recommended with these shared controller devices when there is simultaneous access to the disk, because the disk accesses will prevent proper tape streaming.

HP 7908/HP 7911/HP 7912/HP 7914

These cartridge tape devices do not support the immediate report mode.

AUTHOR

`ct` was developed by HP and AT&T.

SEE ALSO

mkdev(1M), mknod(1M), tcio(1), disk(7), intro(7), mt(7).

NAME

ddfa - HP DTC Device File Access software

DESCRIPTION

The HP **Datacommunications and Terminal Controller** (DTC) **Device File Access** (DDFA) software allows access from HP-UX systems and user-written applications to HP DTCs using standard HP-UX structures. DDFA provides an interface to remote (LAN-connected) DTC ports that is similar to the interface for local mux ports.

The basic principle is that a daemon is created for each configured port based on information in a configuration file (**dp** file). When the daemon is spawned, it takes a pty from the pool and creates a device file with the same major and minor number as the pty slave. The device file is known as the "pseudonym", and user applications use the pseudonym to access the server port using standard HP-UX intrinsics (open(), **close()**, **write()**, etc). The daemon listens on the pty until an application does an **open** (*pseudonym*), then it manages the connection to the server port. The end result is that the server port is addressed via a device file, but the mechanism that makes it happen is transparent to the user. For example, the **lpadmin** command can be used to set up a connection to a printer on a remote server port by using the pseudonym. A second configuration file (**pcf**) contains information to profile the port's behavior.

DDFA consists of the following items:

dp Dedicated Port configuration file. This text file contains the information DDFA needs to set up and manage a connection to a specified DTC port. It contains a one-line entry for each configured DTC port; the line specifies the board and port, the pseudonym, and the path to the Port Configuration File (**pcf**).

The **dp** file is parsed by the Dedicated Port Parser (**dpp**) which spawns an outgoing connection daemon (**ocd**) for each connection specified in the file.

dp is also used by the HP-UX telnet daemon (**telnetd**) to identify incoming connections. In this usage, the daemon negotiates the telnet environment option, and the DTC returns the board and port numbers of the connecting device. **telnetd** then looks up the port and board numbers in the **dp** file and maps them to the pseudonym (see *dp*(4) for more information).

There are two ways to specify a port: explicitly specify its IP address, or specify the IP address of the server, the board, and the port. Alternately, the server's IP address and the TCP service port address of the port can be given.

pcf Port Configuration File. This file is used by DDFA to configure specific port parameters. **pcf** is the generic name of the template file. In practice it is renamed for each port, and the values are altered appropriately for the device attached to the port. The **pcf** is referenced by an entry in the Dedicated Ports file (**dp**). The Dedicated Port Parser (**dpp**) parses the **dp** file and calls the Outbound Connection Daemon (**ocd**) program to spawn a daemon for each valid line in the **dp** file. A valid line is one in which the fourth field is the name of a **pcf**.

dpp Dedicated Port Parser. **dpp parses the Dedicated Ports file** (**dp**) and calls the Outbound Connection Daemon (**ocd**) to spawn a daemon for each valid entry in **dp**. It can be run from the shell or it can be included in **netlinkrc** to automatically run the DDFA software each time the system is booted.

dpp has one mandatory argument, the path to the **dp** file. The other arguments specify an optional log file, the path to **ocd** if it is not the default, and an option to kill existing processes when the **dp** file is parsed. **dpp** can also be run in check mode, where it simply parses the **dp** file and reports any errors.

ocd Outbound Connection Daemon. **ocd** manages the connection and data transfer to the remote DTC port. Normally, it is spawned by the Dedicated Port Parser (**dpp**), but it can be run directly from the shell. If it is run from the shell, the name or IP address of the server or port and the pseudonym (device file name) must be entered on the command line. The board/port number or TCP service port address must be entered if the IP address does not explicitly name a port.

ocd creates a device file for the connection using a file name that is determined by the *pseudonym* argument. If **ocd** encounters an error condition that causes it to exit, it normally removes the device file it created.

If a device file is accidentally deleted, the corresponding **ocd** continues running for at least 30 seconds before it detects the absence of the device file. The **ocd** then writes an error message to **/sys/adm/syslog** and exits.

ocdebug

Outbound Connection Daemon debug mode. **ocdebug** is a special version of **ocd** that contains debug code. **ocdebug** has to be run from the shell with all the arguments that would normally come from the **dp** file entered on the command line. Except for **-d**, the arguments are the same as the **ocd** arguments. The **-d** option specifies the level of debugging messages.

INSTALLATION

DDFA is provided in **update** format, and is installed using the standard HP-UX **update** command (see *update*(1M).

Note that update can install software from a tape or a file whose name has the suffix **.updt**. To install from a file, specify the file name **ddfa.updt** instead of the device, such as **/dev/rct/c1d0s2**. For more information refer to the file **readme.ddfa** which contains installation instructions.

The files, their default directories, and the correct access permissions are shown below. All files should be owned by user **bin**, group **bin**:

```
-r-xr--r-- /etc/dpp
-r-xr--r-- /etc/ocdebug
-r-xr--r-- /etc/ocd
-rw-r--r-- /etc/newconfig/ddfa/pcf
-rw-r--r-- /etc/newconfig/ddfa/dp
-rw-r--r-- /etc/dpp_login.bin
-rw-r--r-- /etc/utmp.dfa
-rw-r--r-- /etc/readme.ddfa
-r--r--r-- /usr/man/man1m.Z/dpp.1m
-r--r--r-- /usr/man/man1m.Z/ocd.1m
-r--r--r-- /usr/man/man1m.Z/ocdebug.1m
-r--r--r-- /usr/man/man4.Z/dp.4
-r--r--r-- /usr/man/man4.Z/pcf.4
-r--r--r-- /usr/man/man7.Z/ddfa.7
```

CONFIGURATION

There are two basic steps to configuring DDFA software:

- Entering information in the **dp** file,
- Entering information in the **pcf** files.

Configuring the dp file

The **dp** file contains one line for each connection that is to be established. A template **dp** file is provided (**/etc/newconfig/ddfa/dp**) which can be edited directly or a separate file created. It is recommended that the file be copied and the copy edited. We suggest you create the directory **/etc/ddfa** to contain **dp** and the pcfs. Note the following points:

- The default file is **/etc/newconfig/ddfa/dp**; if another file name or path is used, this information must be passed as an argument to **dpp** (see *dpp*(1M) for details), whether it is run from the command line or from **netlinkrc**.
- The exact order of the fields in **dp** must be maintained:

  IP_address board/port pseudonym pc_file_path

- Fields can be separated by spaces or tabs.
- Only one entry is allowed per line.

- Comments can be appended using the # character: everything after the # on any given line is ignored by the parser.
- There are three ways to specify the DTC port: by explicitly giving its IP address, by giving the IP address of the server and then specifying the port and board, or by giving the IP address of the server and the TCP service port address of the port.
- It is recommended that the **pcf** and device file (pseudonym) are named appropriately so that both can be easily identified (such as when listing processes with **ps -ef**). For example, the **pcf** could be named **pcf_lp1** and the device file (pseudonym) **ocd_lp1**, or the **pcf** could be named **pcf_dtc1b2p3** and the device file could be named **ocd_dtc1b2p3**.

dp file entries contain the following fields:

IP_address This is the IP address of the DTC being accessed, or the IP address of the port on the DTC.

board/port This field contains the DTC board and port numbers, separated by a / character. The board and port numbers are not checked by **dpp**, but are checked by **ocd**. Valid values are 0 to 7 for the board, and 0 to 31 for the port (these restrictions do not apply if a TCP service port address is given instead).

If the *IP_address* field explicitly defines the DTC port, the value in the port/board field must be **xx/xx** (use **X** or **x**).

If the entry is of the form **xx/***n* where *n* is a decimal number, *n* is taken to be the TCP port address, and this value is used when the connection is established; otherwise, the destination is filled using the formula $(256 \times (32 \times \text{board} + \text{port} + 1) + 23)$.

pseudonym This is the absolute path and name of the device file known to the system and/or the end user application. The device file name is limited to 14 characters. We recommend that the name reflects the connected device, and is similar to the **pcf** name, for example **ocd_dtc1b1p1**.

pc_file_path This is the path to the Port Configuration File (**pcf**) that contains the configuration information for the DTC port. This field is mandatory because the parser **dpp** uses the presence of this field as its flag to spawn a daemon for the line. We recommend that the name of the file reflect the connected device, and be similar to the **pcf** name, for example **pcf_dtc1b1p1**.

The following examples illustrate the structure of typical **dp** file entries.

A printer is connected to port 1 of board 3 of a DTC at IP address 11.234.87.123. The printer attached to the port can be accessed with the HP-UX spooler by specifying device file **/dev/ocd_lp1** in the **lpadmin** command. The port is profiled using data in the file **/etc/ddfa/pcf_lp1**:

```
11.234.87.123 03/01 /dev/ocd_lp1 /etc/ddfa/pcf_lp1 # lp1 b1,n2,f7
```

A printer is connected to the DTC port at IP address 11.234.87.124, so the board/port field contains **xx/xx**. File **pcf_lp2** contains port profiling information.

```
11.234.87.124 xx/xx /dev/ocd_lp2 /etc/ddfa/pcf_lp2 # lp2 b2,n1
```

Specify a port using a TCP port address. The port address is calculated using the formula $(256 \times (32 \times \text{board} + \text{port} + 1) + 23)$.

```
11.234.87.215 xx/16919 /dev/ocd_lp3 /etc/ddfa/pcf_lp3 # lp3 b2,p1
```

Configuring the pcfs

Port Configuration Files (**pcf**) are used to configure individual server ports. **/etc/newconfig/ddfa/pcf** is a template file; in practice it should be renamed for each port, and the values it contains altered to suit the device attached to the port. We recommend you create the directory **/etc/ddfa** to contain the modified pcfs and the modified **dp** file. The pcf is referenced by an entry in the Dedicated Ports file (**dp**). The Dedicated Port Parser (**dpp**) parses the **dp** file and calls the Outbound Connection Daemon (**ocd**) program to spawn a daemon for each valid line in the **dp** file; a valid line is one in which the fourth field is the name of a pcf.

Note the following points:

- The order of the variables is not important.
- The file consists of the names of variables and their values. The variables in the template file are shown terminated by a colon (`:`), but this is not mandatory.
- The variables and their values can be separated by spaces or tabs.
- Only one variable-value pair is allowed per line.
- Only the value should be altered; the variable name must remain the same.
- If you want to use the default values, simply reference the template file `/etc/newconfig/ddfa/pcf` in the `dp` file.
- If you want to use different values for a port, make a copy of `/etc/newconfig/ddfa/pcf`. It is a good idea to name the file appropriately for the pseudonym (device file); for example, `pcf_dtc1b2p3`.

The file contains the following information:

`telnet_mode:` Recognized values are `disable` and `enable`. When enabled, data transfer over the network uses Telnet protocol. This option *must* be enabled for the DTC

`timing_mark:` Recognized values are `enable` and `disable`. When enabled, a telnet timing-mark negotiation is sent to the DTC after all user data has been transferred. `ocd` waits for a reply to the timing mark negotiation before closing the connection. This ensures that all data has been output from the DTC buffers to the device before the buffers are flushed. It should therefore be enabled for the DTC.

`telnet_timer:` Defines the time, in seconds, during which the software waits for a response to the telnet timing mark and binary negotiation. If the timer expires, an error message is logged to `/usr/adm/syslog` and the error is transmitted to the user application.

`binary_mode:` Recognized values are `disable` and `enable`. When it is enabled, data transfer over the network is in binary mode, and treatment of special characters (such as XON/XOFF) is disabled.

Due to the absence of flow control, data integrity cannot be guaranteed when binary_mode is enabled.

Note that even if `binary_mode` is disabled, it can be negotiated at any time by the application setting `IXON` to zero in the termio data structure.

`open_tries:` This defines the number of times the software tries to open a connection before giving up. If the value is `0` the software tries "forever" (approximately 68 years). If the retry process fails, an error message is logged to `/usr/adm/syslog`. The error message is also transmitted to the user application.

The retry process can be interrupted by sending the `SIGUSR2` signal to the `ocd` process by using `kill -17` *pid*.

Note that if the application exits after asking `ocd` to open the connection to the DTC, `ocd` continues trying to open until `open_tries` and/or `open_timer` are exceeded.

`open_timer:` Defines the time, in seconds, between tries. If the value is `0`, `ocd` uses an exponential retry period algorithm up to 32 seconds; that is, 1 2 4 8 16 32 32 32 ...

`close_timer:` Defines the time, in seconds, between the close call made by the application on the pty slave and the moment when the connection is actually closed. Setting this value to, for example, 5 seconds avoids the overhead of opening and closing the connection when a spooler spools several files at a time. Setting a sufficiently high value effectively leaves the connection permanently open.

`status_request:` Recognized values are `disable` and `enable`. When enabled, the software sends a status request to the printer attached to the server and processes the reply as

follows:

LP_OK	(0x30)	ocd continues processing.
LP_NO_PAPER	(0x31)	ocd retries within the limits of the status timer.
LP_BUSY	(0x32)	ocd retries within the limits of the status timer.
LP_OFF_LINE	(0x23)	ocd retries within the limits of the status timer.
LP_DATA_ERROR	(0x38)	ocd retries within the limits of the status timer.

status_timer: Defines the time, in seconds, after which the software no longer waits for the reply to the status request. If the timer expires, an error message is logged to /usr/adm/syslog. The error condition is also transmitted to the user application.

eight_bit: Recognized values are **enable** and **disable**.

Normally, data bytes processed by the pty have bit 7 stripped. If **eight_bit** is enabled, the stripping is disabled. If **eight_bit** is disabled, stripping is enabled, and bit 7 is stripped. This can also be achieved by changing the pseudonym's termio structure using the **ioctl()** commands.

tcp_nodelay: Recognized values are **enable** and **disable**.

If enabled, data is sent to the LAN as it is received. It can be disabled if the software is sending packets faster than the server can accept.

Default values are:

```
telnet_mode: enable
timing_mark: enable
telnet_timer: 120
binary_mode: disable
open_tries: 1500
open_timer: 30
close_timer: 0
status_request: disable
status_timer: 30
eight_bit: disable
tcp_nodelay: enable
```

Configuring netlinkrc

DDFA can be run at boot time by including a reference to **dpp** in **netlinkrc**. **dpp** is run, the **dp** file is read, and the appropriate processes created. It is recommended that the **-k** option be used, for example:

```
dpp:2:once:/etc/dpp /etc/ddfa/dp_file -k
```

ERROR HANDLING

When **ocd** receives a serious error condition, such as when the LAN goes down, **ocd** transmits the error conditions to the application by closing the pty. Any **open()**, **close()**, or **write()** to the pseudonym returns the error condition **0 bytes read** . If the pseudonym is the controlling terminal for the group to which the application belongs, **sighup** is sent to all the processes in the group, including the application.

KILLING DAEMONS

Outbound Connection Daemons (**ocd**s) should be killed using **kill -15**. **kill -9** is not recommended because the device file remains open. **ocd** verifies the validity of an existing pseudonym before trying to use it. **dpp** and *ocd* use data stored in file **/etc/utmp.dfa** to verify whether a process still owns a pseudonym before taking it over. If **ocd** finds an unowned pseudonym, it will use it.

ioctl() LIMITATIONS

Not all ioctl functionality is available, due to the lack of a protocol that allows the transmission of such commands over the LAN to the remote port.

TERMIO Attribute Limitations

The main restrictions include modem signal control and parity checking. The following are not available:

IGNPAR	INPCK	IXOFF
PARMRK	IXANY	CBAUD

ioctl() Request Limitations

The following `ioctl()` request limitations apply:

`TERMIO` structure:

`CSTOPB` flag	DTC only supports one stop bit.
`CSIZE`	DTC only supports 8 bits per character. Value cannot be modified.
`PARODD` flag	DTC offers static configuration to handle even or odd parity. It also handles auto parity detection for even or odd parity.
`PARENB` flags	Enabling/disabling done via static configuration. No programatic interface supplied.
`INPCK` flag	No way to separate input from output parity features.
`IGNPAR` flag	Cannot be configured on DTC.
`PARMRK`	Bad characters are forwarded to the system without marking them with OFFH or OH.
`CBAUD`	Speed is part of static configuration.
`IXOFF` flag	Flow control is enabled if the DTC static configuration specifies an ASCII access mode. If binary is selected, no flow control is provided.
`IXON` flags	Pacing of output to a terminal via a programmatic interface is enabed when ASCII mode is selected in static port configuration and disabled when binary mode is selected.
`IXANY` flag	DTC does not offer ability to restart output on any character received if XOFF was previously received.
`HUPCL` flag	DDFA does not support the hanging up of modem signals on the last close of the device file. If the modem signals used on the DTC drop, the connection is closed.
`CLOCAL` flag	Not supported.
`c_flags`:	`IENQACK` not supported. `OFILL`, `OFDEL`, `NLDLY`, `CRDLY`, `TABDLY`, `BSDLY`, `FFDLY` not supported by TELNETD-DTC software.
`BINARY` mode flags	Part of static configuration is done in DTC Manager by selecting binary mode. If switching is enabled, binary can be selected at user interface level. There is no way to automatically negotiate binary mode when proper termio flags are reset when using `telnetd`. Binary/ASCII switching is possible with DDFA. The DTC cannot support large reads in pure binary mode, so transferred blocks of data should not be more than 256 bytes. If half-duplex with remote acknowledgement is implemented, binary applications can be supported.

ioctl() System Call Requests

The following `ioctl()` system call limitations apply:

`TCSBRK` The ability to send a break without waiting for previous data to be sent is not provided at system level in `telnetd` or DDFA. Receiving a telnet break command in the DTC allows it to generate a break on asynchronous ports.

`TCFLSH` The DTC output queue cannot be flushed.

Hardware handshake request
Not supported on DTC.

`TCXONC` Local handshake cannot be disabled on DTC.

MCGETA Not supported.

MCSETA, MCSETAF, MCSETAW
There is no way to separately set modem lines of a DTC port.

MCGETT Modem timers, CD timer, connect timer, and disconnect cannot be configured.

CCITT simple, and direct call-in/call-out modes
DTC cannot handle simple mode because there is programmatic interface for modem signals. Call-in mode cannot be simulated if the port is opened, because modem signals (or the call) must be present within 2 minutes or the connection is cleared.

DACIDY get device adapter info
No way to get device adapter information.

download **ioctl DACRADDR, DACDLADDR, DACDLGO, DACDLVER**
No programmatic call to download the DTC.

DACHWSTATUS, DACSELFTEST, DACLOADED, DACISBROKE status
No programmatic interface to get such info.

DACLOOPBACK DACSUBTEST port test

FILES

```
/etc/dpp
/etc/ocdebug
/etc/ocd
/etc/dpp_login.bin
/etc/utmp.dfa
/etc/newconfig/ddfa/pcf
/etc/newconfig/ddfa/dp
```

SEE ALSO
dpp(1M), ocd(1M), ocdebug(1M), ioctl(2), dp(4), pcf(4) ioctl(5).

NAME

diag0 - diagnostic interface to I/O subsystem

DESCRIPTION

`diag0` is the diagnostic pseudo-driver used by the on-line diagnostic subsystem. The kernel sends diagnostic events here for logging by the diagnostic subsystem. The diagnostic subsystem also sends diagnostic requests to the kernel via `diag0`.

WARNINGS

If the diagnostic subsystem does not read diagnostic events from `diag0` as fast as they are logged by the kernel, a warning message `Warning: DIAG0 -- message queue full` appears on the console.

AUTHOR

`diag0` was developed by HP.

FILES

`/hp-ux`	
`/dev/diag0`	
`/dev/diag/diag0`	
`/dev/diag`	directory containing diagnostic device files

SEE ALSO

sysdiag (1M), diaginit (1M).

Online Diagnostic Subsystem manuals.

NAME

disk - direct disk access

DESCRIPTION

This entry describes the actions of HP-UX disk drivers when referring to a disk as either a block-special or character-special (raw) device.

Device File Naming Conventions

Standard disk device files are named according to the following conventions:

Block-mode Devices `/dev/dsk/[r][cxd]y[ln]sm`

Character-mode Devices `/dev/rdsk/[r][cxd]y[ln]sm`

where component parts of the filename are constructed as follows:

`r`	Do not use. Reserved for future use.
`c`	Optional. Identifies the following hexadecimal digits as a non-default "controller number".
x	Hexadecimal number identifying a non-default controlling bus interface. Required if `c` is specified.
`d`	Required if `c` is specified. Identifies the following hexadecimal digits as a "drive number".
y	Hexadecimal number identifying the drive number (bus address). Required regardless of whether `cxd` is specified.
`l`	(ell) Optional. Identifies the following hexadecimal digits as a "logical unit number".
n	Hexadecimal logical unit number of the device. Required if `l` is specified.
`s`	Required. Identifies the following value as a "section number".
m	Required. Drive section number.

If `cxd` is not specified, the default controlling bus interface is used. Some administrators may prefer to completely specify `cxd` to eliminate any ambiguity in file naming versus device and system configuration.

Assignment of controller, drive, logical unit and section numbers is described in the system administrator manuals for your system.

Block-special access

Block-special device files access disks via the system's block buffer cache mechanism. Buffering is done in such a way that concurrent access through multiple opens and mounting the same physical device is correctly handled to avoid operation sequencing errors. The block buffer cache permits the system to do physical I/O operations when convenient. This means that physical write operations may occur substantially later in time than their corresponding logical write requests. This also means that physical read operations may occur substantially earlier in time than their corresponding logical read requests.

Block-special files can be read and written without regard to physical disk records. Block-special file `read()` and `write()` calls requiring disk access result in one or more `BLKDEV_IOSIZE` byte (typically 2048 byte) transfers between the disk and the block buffer cache. Applications using the block-special device should ensure that they do not read or write past the end of last `BLKDEV_IOSIZE` sized block in the device file. Because the interface is buffered, accesses past this point behave unpredictably.

Character-special access

Character-special device files access disks without buffering and support the direct transmission of data between the disk and the user's read or write buffer. Disk access through the character special file interface causes all physical I/O operations to be completed before control returns from the call. A single read or write operation up to `MAXPHYS` bytes (typically 64 Kbytes or 256 Kbytes) results in exactly one disk operation. Requests larger than this are broken up automatically by the operating system. Since large I/O operations via character-special files avoid block buffer cache handling and result in fewer disk operations, they are typically more efficient than similar block-special file operations.

There may be implementation-dependent restrictions on the alignment of the user buffer in memory for character special file `read()` and `write()` calls. Also, each read and write operation must begin and end on a sector boundary and must be a whole number of sectors in size. The sector size is a hardware-dependent value that can be queried with the `DIOC_DESCRIBE` ioctl call, which is described below.

In addition to reading and writing data, the character-special file interface can used to obtain device specific information and to perform special operations. These operations are controlled through use of ioctl calls. Details related to these ioctls are contained in <sys/diskio.h>.

The DIOC_DESCRIBE ioctl can be used to obtain device specific identification information. The information returned includes the disk's model identification, the disk interface type, and the disk's sector size.

The DIOC_CAPACITY ioctl can be used to obtain the capacity of a disk device in DEV_BSIZE units. (DEV_BSIZE is defined in <sys/param.h>).

The DIOC_EXCLUSIVE ioctl can be used to obtain and release exclusive access to a disk device. Exclusive access is required for some special operations, such as media reformatting, and may be desirable in other circumstances. The value one specifies that exclusive access is requested. The value zero specifies the exclusive access should be released. Exclusive access causes other open requests to fail. Exclusive access can only be granted when the device is not currently opened in block-mode and there is only one open file table entry for that disk device (the one accessible to the exclusive access requester).

ERRORS

The following errors can be returned by a disk device driver call:

[EACCES]	Required permission is denied for the the device or operation.
[ENXIO]	If resulting from an open() call, this indicates there is no device at the specified address. For other calls, this indicates the specified address is out of range or the device can no longer be accessed.
[EINVAL]	From an open() call: the device is not a disk device. For other calls: Invalid request or parameter.
[EIO]	I/O error (e.g., media defect or device communication problem).

WARNING

The interaction of block-special and character-special file access to the same BLKDEV_IOSIZE-sized block is not specified, and in general is unpredictable.

On some systems, having both a mounted file system and a block special file open on the same device can cause unpredictable results; this should be avoided if possible. This is because it may be possible for some files to have private buffers in some systems.

Although disk devices have historically had small (typically 512-byte) block sizes, some disk devices (such as optical disks and disk arrays) have relatively large block sizes. Applications using direct raw disk access should use ioctl() calls to determine appropriate I/O operation sizes and alignments.

DEPENDENCIES

Series 800

Devices whose sector size is less than DEV_BSIZE must be accessed on DEV_BSIZE boundaries and with transfer sizes that are multiples of DEV_BSIZE.

AUTHOR

disk was developed by HP and AT&T.

SEE ALSO

mkdev(1M), mknod(1M), ct(7), intro(7).

System Administrator manuals included with your system.

NAME

floppy - direct flexible or "floppy" disk access

DESCRIPTION

Flexible disk devices are removable-media disk devices which are typically used to share data with other systems. Media types are identified by physical size (such as 3.5-inch and 5.25-inch), number of data surfaces (or sides), and data density. By convention, flexible disk devices are named using the same conventions as those used for other disk devices (see *disk*(7)), with the exception that the device special files reside in the directories: **/dev/rfloppy/** and **/dev/floppy/**.

Data can be stored on flexible disk media in a variety of logical formats. The capacity of these devices is generally too small to hold useful HP-UX file systems. Instead, DOS or LIF file systems (see *dosif*(4) and *lif*(4) for a detailed description of these file systems) are commonly used. Data can also be stored in an archive-utility format. For example, **tar** and **cpio** are commonly used to share data with other HP-UX systems (see *tar*(1) and *cpio*(1)).

In addition to the various logical formats, data can be stored on flexible disk media in a variety of physical data formats called geometries. The following parameters are used to describe a flexible disk geometry:

heads Number of surfaces (or sides) on the media that contain valid data.

tracks Number of tracks on a single media surface or side (the term cylinders is sometimes also used). This value does not include spare tracks.

sectors Number of sectors in a single track. The number of sectors that can fit on a track depends on the bit density (as controlled by transfer rate and media rotation rate) and the sector size.

sector size Number of bytes in a logical sector. Since all I/O operations must be an integral number of sectors in length, this parameter also indicates the minimum character-special file I/O size.

transfer rate Media data rate in Kbits per second. The transfer rate is an indirect means of representing bit density. Bit density is measured in bits per radian, and is the formal intra-track data density parameter for standard specification. Transfer rate is generally used to program flexible media devices and is therefore more appropriate for this interface. Since the media rotation rate for most flexible disk devices is standard, conversion between these two representations is straight-forward.

track density Number of tracks per inch. Some low density formats can be supported on high-density drives by skipping tracks during head stepping.

data encoding Encoding method used to store data. FM (frequency modulation) and MFM (modified frequency modulation) are the most common encoding methods.

The following table shows some useful flexible disk media geometries (without density information). The right-most column indicates which **mediainit -f** option should be used to format media to the indicated geometry (see *mediainit*(1M)).

Media Type	Use	Capacity	Heads	Tracks	Sectors	Sector Size	-f
3.5in DS DD		630,784	2	77	16	256	1
3.5in DS DD		655,360	2	80	16	256	21
3.5in DS DD		709,632	2	77	9	512	2
3.5in DS DD	DOS 720K	737,280	2	80	9	512	16
3.5in DS DD		788,480	2	77	5	1024	3
3.5in DS HD		1,261,568	2	77	32	256	1
3.5in DS HD		1,419,264	2	77	18	512	2
3.5in DS HD	DOS 1.44M	1,474,560	2	80	18	512	16
3.5in DS HD		1,567,960	2	77	10	1024	3
3.5in DS HD		1,638,400	2	80	10	1024	23
5.25in DS	DOS 360K	368,640	2	40	9	512	2
5.25in DS HD	DOS 1.2M	1,228,800	2	80	15	512	16

The following table shows the density information for some standard flexible disk media.

MediaType	BitDensity	RPM	TransferRate	TrackDensity	DataEncoding
3.5in DS DD	7,958	300	250	135	MFM
3.5in DS HD	15,916	300	500	135	MFM
5.25in DS	7,958	300	250	48	MFM
5.25in DS HD	13,262	360	500	96	MFM

Abbreviations are interpreted as follows:

DS Double-sided media
DD Double-density media
HD High-density media

Normally each **open()** call causes the device and/or floppy device driver to attempt to determine the geometry of the installed media. As a result, the current flexible disk geometry is set to the supported geometry that matches the physical data format on the media currently installed in the device. To maintain reasonable open times, not all possible media geometries are checked. Therefore, it is possible that a flexible disk medium may contain valid data even though its format is not recognized. This automatic geometry sensing functionality may be disabled in some drivers by use of the **O_NDELAY** flag in the **open()** call or device driver dependent minor numbers.

The **FLOPPY_GET_INFO** ioctl indicates the characteristics and current status of a floppy device. Information for interpreting the **media** and **data_encoding** fields can be found in <sys/floppy.h>. The following macros are defined for decoding the **status** and **valid** fields. These macros return a non-zero (true) value for the associated conditions.

```
FLOPPY_NO_MEDIA(x)          /* Drive is empty */
FLOPPY_BLANK_MEDIA(x)       /* Media geometry is not recognizable */
FLOPPY_WRITE_PROT(x)        /* Media is write protected */
FLOPPY_MEDIA_CHANGED(x)     /* Media has changed since last status */
FLOPPY_HIGH_DENSITY(x)      /* Media has high density indication */
```

Some floppy devices or floppy device drivers may be unable to determine some status information. The **valid** field indicates whether or not the corresponding status information is meaningful. Applying a macro to the **valid** field indicates whether or not the application of that same macro to the **status** field results in a valid device status.

The **FLOPPY_GET_GEOMETRY** ioctl can be used to determine the flexible disk device's current media geometry. Current geometry parameters are indicated in the fields of the resultant **floppy_geometry** structure.

The **FLOPPY_SET_GEOMETRY** ioctl can be used to specify the desired media geometry. Exclusive access to the device, obtained through use of the **DIOC_EXCLUSIVE** ioctl (see *disk*(7)), is required prior to setting the media geometry. Exclusive access is necessary to ensure that other applications are not affected.

The **FLOPPY_FORMAT_TRACK** ioctl can be used to reformat a media track. Exclusive access to the device, obtained through use of the **DIOC_EXCLUSIVE** ioctl (see *disk(7)),* is required prior to reformatting to ensure that other applications are not affected.

Flexible disk devices support the generic disk ioctls (see *disk*(7)), typically used for hard disk devices. Flexible disk device drivers may also support driver specific ioctls (see the appropriate driver manual section).

The header file **<sys/floppy.h>** has useful information for flexible-media device control. The following is included from **<sys/floppy.h>**:

```
/* ioctls for flexible (floppy) disk devices */
#define FLOPPY_GET_INFO          _IOR('F', 1, struct floppy_info)
#define FLOPPY_GET_GEOMETRY      _IOR('F', 2, struct floppy_geom)
#define FLOPPY_SET_GEOMETRY      _IOW('F', 3, struct floppy_geom)
#define FLOPPY_FORMAT_TRACK      _IOW('F', 4, struct floppy_format)

/* structure for FLOPPY_GET_STATUS ioctl */
struct floppy_info {
```

```
        unsigned        media;
        unsigned        status;
        unsigned        valid;
};
/* structure for FLOPPY_GET_GEOMETRY and FLOPPY_SET_GEOMETRY ioctls */
struct floppy_geometry {
        unsigned        heads;
        unsigned        tracks;
        unsigned        sectors;
        unsigned        sector_size;
        unsigned        transfer_rate;
        unsigned        track_density;
        unsigned        data_encoding;
};
/* structure for FLOPPY_FORMAT ioctl */
struct floppy_format {
        unsigned        head;
        unsigned        track;
        unsigned        interleave;
};
```

ERRORS

The following errors can be returned by a flexible-disk device-driver call:

[EACCES] Required permission is denied for the the device or operation.

[ENXIO] If resulting from an open call, this indicates there is no device at the specified address. For other calls, this indicates the specified address is out of range or the device can no longer be accessed.

[EINVAL] From an `open()` call: the device is not a floppy device. For other calls: Invalid request or parameter.

[EIO] I/O error (e.g., media defect or device communication problem).

WARNING

A floppy disk containing a mounted file system should not be removed prior to being unmounted. Removal of floppy disks containing mounted file systems is likely to result in file system errors, and system panics.

Reformatting flexible disk media from one geometry to another that differs only in that it has fewer tracks can cause the automatic geometry sensing functionality of `open()` to fail to recognize the new media geometry. Bulk erasing (degaussing) the media or formatting the media to a substantially different geometry prior to reformatting prevents automatic geometry sensing problems.

Single track formatting may not be supported by some floppy devices.

Some devices permit configuration for geometries which they are unable to support. The FLOPPY_SET_GEOMETRY ioctl for such a configuration may not result in an error. However, subsequent I/O operations will fail.

DEPENDENCIES

Devices and Drivers

Geometry support varies, depending on device and device driver capabilities.

SEE ALSO

disk(7), mediainit(1M), mknod(1M), dosif(4), lif(4).

NAME

framebuf - information for raster frame-buffer devices

SYNOPSIS

#include <sys/framebuf.h>

DESCRIPTION

Frame-buffer devices are raster-based displays. These devices use memory-mapped I/O to obtain much higher performance than possible with tty-based graphic terminals. Frame-buffer devices can be accessed directly using this interface, although access through the STARBASE libraries is recommended (see *starbase*(3G)). Direct access to frame-buffer devices entails precise knowledge of the frame-buffer architecture being used. Input cannot be piped into or redirected to frame-buffer devices because they are not serial devices.

Each frame-buffer device is associated with a character special file. Major and minor numbers for frame-buffer devices are implementation-dependent. The minor numbers for these devices denote different frame buffers. Implementation-specific details are discussed in the appropriate systems administrator's manuals.

Communication with a frame-buffer device begins with an *open*(2) system call. Multiple processes can have the frame-buffer device open concurrently.

close(2) invalidates the file descriptor associated with the frame-buffer device. After a *close* system call, any access to the frame-buffer device address range might result in a memory fault and the signal SIGSEGV being sent to the process (see *signal*(2)). A process cannot unmap the frame buffer from its address space after the frame-buffer special file is closed. To unmap a frame buffer, use the GCUNMAP *ioctl*(2) call (see below).

Once a process acquires a lock for the frame-buffer device, it must unlock it explicitly before calling *close*(2); see GCUNLOCK below.

read(2) and *write*(2) system calls are undefined and always return an error. In this case **errno** is set to ENODEV.

The *ioctl*(2) system call is used to control a frame-buffer device. The *select*(2) system call is used to test the frame-buffer device for exceptional conditions. Interrupts from the graphic hardware are considered exceptional conditions. An exceptional condition is automatically cleared after any process that opens the frame-buffer device is notified of the exception by a *select*(2) call. A call to *select*(2) for read or write on the file descriptor associated with the frame-buffer device returns a false condition in the read and write bit masks (see *select*(2)).

A frame-buffer device can be accessed by multiple processes at once. However, each process overwrites the output of the others unless one of the lock mechanisms described here or some other synchronization mechanism is used. The lock mechanisms described here are intended for cooperating processes only.

For all frame buffers, data bytes scan from left to right and from top to bottom. A pixel, which is a visible dot on the screen, is associated with a location in the frame buffer. Each device maps one or more bits in memory to a pixel on the screen, although the bits in the frame buffer might not be continuous. Information describing the frame-buffer structure and attributes is found in the **crt_frame_buffer_t** data structure. The **crt_frame_buffer_t** data structure includes the following fields:

```
int crt_id;                     /*display identifier*/
unsigned int crt_attributes;    /*flags denoting attributes*/

char *crt_frame_base;           /*first byte in frame-buffer memory*/
char *crt_control_base;         /*first byte of the control*/
                                /*registers*/

char *crt_region [ CRT_MAX_REGIONS ];
                                /*other regions associated with the*/
                                /*frame-buffer device*/
```

The following are valid *ioctl*(2) requests:

GCDESCRIBE
: Describe the size, characteristics, and mapped regions of the frame buffer. The information is returned to the calling process in a **crt_frame_buffer_t** data structure, and the parameter is defined as **crt_frame_buffer_t * arg ;**. Although some structure fields contain addresses of one or more frame-buffer device regions, the values of these fields are not always defined. Only after a successful GCMAP command is issued (see below) are the correct addresses returned so the user can access the frame-buffer regions directly using the returned addresses.

GCID
: Provide a device identification number. The parameter is defined as **int** **arg*;. The information returned when using this command is a subset of the information provided by GCDESCRIBE, and is provided here for backward compatibility only.

GCON,
: GCOFF Turn graphics on or off. These operations are valid for devices whose CRT_GRAPHICS_ON_OFF bit is set in the **crt_attributes** field of the **crt_frame_buffer_t** data structure returned by the GCDESCRIBE command. Otherwise, these commands have no effect.

GCAON,
: GCAOFF Turn alpha on or off. These operations are valid for devices whose CRT_ALPHA_ON_OFF bit is set in the **crt_attributes** field of the **crt_frame_buffer_t** data structure returned by the GCDESCRIBE command. Otherwise, these commands have no effect.

GCMAP
: Make the frame-buffer memory, graphics control, and other device regions accessible to the user process making the call. Only processes that request this can directly access frame-buffer memory and control registers. After a successful GCMAP call, the fields **crt_frame_base** and **crt_control_base** in the **crt_frame_buffer_t** data structure (returned by a subsequent GCDESCRIBE *ioctl*(2) call), hold the valid addresses of these two regions of the frame buffer. If, for a specific device, more than two regions are to be mapped to the user's address space, the base addresses of up to CRT_MAX_REGIONS extra device regions will be placed in the array **crt_region** in successive order. Only the regions pertinent to a specific frame buffer are mapped. Irrelevent region fields in the **crt_frame_buffer_t** data structure are set to **0**. Use of the *arg* parameter is implementation dependent (see DEPENDENCIES below). The base addresses for frame-buffer regions are always page aligned.

GCUNMAP
: Cause access to the frame-buffer memory, graphics control, and possibly other device regions to be removed from the requesting process. The parameter *arg* is ignored and should be set to **0**. Any attempt to access these memory regions after a successful GCUNMAP call results in a memory fault and sends the signal SIGSEGV to the process.

GCLOCK
: Provide for exclusive use of the frame-buffer device by cooperating processes. The calling process either locks the device and continues or is blocked. Blocking in this case means that the call returns only when the frame buffer is available or when the call is interrupted by a signal. If the call is interrupted, it returns an error and **errno** is set to EINTR. Waiting occurs if another process has previously locked this frame buffer using the GCLOCK command and has not executed a GCUNLOCK command yet. The GCLOCK command does not prevent other non-cooperating processes from writing to the frame buffer; thus, GCLOCK is an advisory lock only. The parameter *arg* is ignored and should be set to **0**.

  This call prevents the Internal Terminal Emulator (ITE) from corrupting the state of the graphics hardware (see *termio*(7)). On some systems, as long as the frame buffer is locked with a GCLOCK command, the ITE does not output text to it (see DEPENDENCIES below). Any attempt to lock the device more than once by the same process fails, and causes **errno** to be set to EBUSY.

GCLOCK_NOWAIT
: Provide for exclusive use of the frame-buffer device by cooperating processes. This request has the same effect on the frame-buffer device as does the GCLOCK request. However, this call does not wait for the frame buffer to be released by other processes. If the frame-buffer device is locked, the process is not blocked; instead, the system call returns an error and causes **errno** to be set to EAGAIN. The parameter *arg* is ignored and should be set to **0**.

GCLOCK_BLOCKSIG Provide for exclusive use of the frame-buffer device by cooperating processes while blocking all incoming signals for the calling process that otherwise might have been caught. This call is a superset of the GCLOCK call. The parameter *arg* is ignored and should be set to **0**. When the display is acquired for exclusive use (and thus locked), all signals sent to the process that otherwise would have been caught by the process *at the time of the* GCLOCK *call*, are withheld (blocked) until GCUNLOCK is requested. Any attempt to modify the signal mask of the process (see *sigsetmask*(2)) before a GCUNLOCK request is made will not have any effect on these blocked signals. The signals are not blocked until the lock is actually acquired, and might be received while still awaiting the lock.

The signal SIGTSTP is also blocked whether or not it is being caught. The signals SIGTTIN and SIGTTOU are also blocked on frame-buffer devices where the ITE does not output to the device while it is locked. See DEPENDENCIES below.

Except for the three signals mentioned above, this call does not block signals that the process did not expect to catch, nor does it block signals that cannot be caught or ignored. This command does not prevent other non-cooperating processes from writing to the frame buffer.

GCLOCK_BLOCKSIG_NOWAIT
Provide for exclusive use of the frame-buffer device by cooperating processes, while blocking all incoming signals for the calling process that otherwise would have been caught. This request has the same effect on the frame-buffer device as does the GCLOCK_BLOCKSIG request. However, this call does not wait for the frame buffer to be released by other processes. If the frame-buffer device is locked, the process is not blocked, but the system call returns an error and causes **errno** to be set to EAGAIN. The parameter *arg* is ignored and should be set to **0**.

GCUNLOCK Relinquish exclusive use of the frame-buffer device. If the device is locked with a GCLOCK_BLOCKSIG or GCLOCK_BLOCKSIG_NOWAIT *ioctl*(2) request, the signal mask of the calling process is restored to its state prior to the locking request.

GCRESET Reset the graphic hardware associated with the frame-buffer device to a defined initial state. The call enables the frame-buffer device to respond to the *ioctl* requests defined here.

GCDMA_OUTPUT Send DMA output to the frame-buffer device. This system call is used to transfer data from a user's array to a rectangular area of the graphics frame-buffer, or optionally, to the device's graphics control space.

The parameters for the DMA are passed in a "crt_dma_ctrl_t" data structure, which includes the following fields:

```
char *mem_addr;     /* Starting address of data
                       being transferred */
char *fb_addr;      /* Address of framebuffer
                       destination */
int length;         /* Number of bytes to transfer,
                       including those "skipped" */
int linelength;     /* Number of bytes written
                       on each framebuffer row */
int skipcount;      /* Number of source bytes to
                       ignore after each "linelength" */
unsigned int flags; /* Specified options to the driver */
```

To write to the graphics frame-buffer, set **fb_addr** to the address of the upper-left corner of the rectangle to be drawn. The DMA will write **linelength** bytes on each frame-buffer row, ignore the next **skipcount** bytes of memory data, then resume writing at the same starting position on each succeeding frame-buffer row. This is continued until **length** bytes are either written or ignored.

To write to the graphics control space, set **fb_addr** to the address of the first graphics control register to write. In this case, **linelength** and **skipcount** are ignored.

The **flags** parameter specifies options for the DMA. Currently, there are no supported flags and this parameter should be set to zero, otherwise the system call will fail and **errno** is set to EINVAL.

The DMA has the same effect on the frame-buffer device as using store instructions to write the data. Thus, various graphics control registers may affect the results of the DMA. It is the responsibility of the user program to perform any necessary set-up of the frame-buffer device so that the DMA has the desired results.

The **skipcount** parameter allows the user to refresh a portion of a window image that the user has stored in memory for those cases where only a portion of the image needs to be refreshed. The window image is then a superset of the rectangle being updated, and might thus have different dimensions. The **skipcount** specifies the portion of the row in the larger window image that is excluded from the rectangle. Thus, **linelength** plus **skipcount** would be the number of bytes in each row of the larger window image array.

If a particular framebuffer device supports this system call, the CRT_DMA_OUTPUT flag in the **crt_attributes** field of the **crt_frame_buffer_t** structure is set. Some framebuffer devices supporting DMA might restrict alignment of the various parameters, and are specified in the DEPENDENCIES section below. The kernel ensures that these restrictions are obeyed, and if they are not the system call will fail and set **errno** to EINVAL.

It is the responsibility of the application to guarantee that the system's physical memory is up-to-date by flushing the processor's data cache. One should use the GCDMA_DATAFLUSH ioctl to ensure that the data is consistent before initiating a DMA transfer.

GCDMA_DATAFLUSH Flush the specified data from the processor's data cache to the system's main memory. This system call is intended to be used before DMA to ensure that an up-to-date version of the data is transferred to the framebuffer or to control space.

The parameters for the flush are passed in a **crt_flush_t** data structure, which includes the following fields:

```
char *flush_addr;    /* Starting address of data
                        to be flushed */
int flush_len;       /* Number of bytes to flush */
```

The kernel ensures that the **flush_len** bytes starting at **flush_addr** are consistent in main memory with respect to the cache.

GCSLOT Provide pertinent information about the calling process's participation in the system-wide graphics locking mechanism (see the discussion under GCLOCK above). The GCSLOT request does not carry out any actual locking functionality. The lock information is returned to the calling process in a **crt_gcslot_t** data structure. The parameter is defined as **crt_gcslot_t *arg;**. The **crt_gcslot_t** data structure is defined in the file **<sys/framebuf.h>**.

GCSTATIC_MAP Prevent the Internal Terminal Emulator (ITE) from modifying the device's color map.

GCVARIABLE_MAP Allow the Internal Terminal Emulator (ITE) to modify the device's color map.

DEPENDENCIES

Series 700/800

When requesting GCMAP, the parameter *arg* is ignored and should be set to **0**.

All supported ITEs ignore the frame buffer lock for output.

Series 700

Among the device identification constants that can be returned both by the GCID call and in the **crt_id** field of the **crt_frame_buffer_t** data structure by the GCDESCRIBE call are:

S9000_ID_98705
S9000_ID_98736
S9000_ID_A1659A
S9000_ID_A1439A

If a memory-mapped graphics co-processor is available, it is mapped in with other graphics regions as the result of a GCMAP call, and its addres is recorded as the last entry in the crt region array returned by the GCDESCRIBE call.

Series 800

The following device identification constants are returned both by the GCID call and in the **crt_id** field of the **crt_frame_buffer_t** data structure by the GCDESCRIBE call:

S9000_ID_98720
S9000_ID_98730
S9000_ID_98550

For the HP A1047A Interface Card, the fields of the **crt_dma_info** structure have the following restrictions:

mem_addr	32-byte aligned
fb_addr	16-byte aligned
length	non-zero multiple of 32
skipcount	0

ERRORS

[EAGAIN]	The operation would result in suspension of the calling process, but the request was either GCLOCK_NOWAIT or GCLOCK_BLOCKSIG_NOWAIT.
[EBUSY]	Attempted to lock the device, which is already locked by the same process.
[EINTR]	A call to *ioctl*(2) was interrupted by a signal.
[EINVAL]	An invalid *ioctl*(2) command was made.
[ENODEV]	Attempted to use *read*(2) or *write*(2) system calls on the device.
[ENOMEM]	Sufficient memory for mapping could not be allocated.
[ENOSPC]	Required resources for mapping could not be allocated.
[ENXIO]	The minor number on the device file refers to a nonexistent device.
[EPERM]	Requested GCUNLOCK *ioctl*(2) command, but the device was locked by a different process.

AUTHOR

framebuf was developed by HP.

SEE ALSO

select(2), open(2), close(2), signal(2), sigsetmask(2), lockf(2), ioctl(2), mknod(1M), starbase(3G), termio(7).

NAME

gpio - general-purpose I/O interface

SYNOPSIS

#include <sys/gpio.h>

DESCRIPTION

gpio is a general-purpose I/O interface supporting high-speed parallel communication with an arbitrary peripheral. It includes sixteen data lines, two handshake lines for transfer protocol, a peripheral-controlled interrupt line, and several lines for application-dependent control and status. This section describes the use of the *gpio* driver in the HP-UX system.

Device Attributes

GPIO attributes are classified into two groups: per-open, and per-interface. File descriptors obtained from separate *open*(2) requests have separate per-open attributes; changing an attribute from the per-open group affects requests on that file descriptor only. Attributes in the per-interface group are shared by all file descriptors on that interface; changing an attribute from one file descriptor affects all users of the interface.

The per-open set of attributes, and the driver requests to change them, are listed in the following table. All other attributes are per-interface.

Attribute	Driver Request
timeout	GPIO_TIMEOUT
signal mask	GPIO_SIGNAL_MASK
lock count	GPIO_LOCK

Transfer Requests

Standard *read*(2) and *write*(2) requests are used for data transfer over *gpio*.

Control Requests

A user can configure the *gpio* driver by using *ioctl*(2) calls:

```
struct io_ctl_status {
  int type;
  int arg[3];
} gpio_control;

ioctl(fildes, IO_CONTROL, &gpio_control);
```

In the **io_ctl_status** structure, the **type** member specifies the type of control function required. The **arg[ ]** array holds any associated arguments. The defined values for **type** and their use are as follows:

GPIO_TIMEOUT

Set timeout. If any DMA transaction for this file takes longer than **arg[0]** microseconds, it aborts with a status of ETIMEDOUT returned to the user. This is used mainly for detecting device failure. A timeout of 0 is equivalent to an hour for HP 27114A/B interface cards, and infinity (that is, no transaction times out) for the HP 28651A interface.

GPIO_WIDTH Set the width of the interface. This request specifies the number of valid data lines on transfers; **arg[0]** holds the desired interface width in bits. All future read requests inspect only the least significant **arg[0]** data lines, and all future writes present data on only those lines. The state of all other data lines is indeterminate. Widths of 8 and 16 bits are supported. If the 16-bit data width is selected, the number of bytes to transfer must be even; otherwise, an error code of EFAULT is returned to the user.

GPIO_SIGNAL_MASK

Define events that cause a signal to be sent to the calling process. Each request overwrites the previous mask for the **fildes**; thus events can be disabled by using a zero.

The value of **arg[0]** defines which events allow the calling process to receive a signal should an asynchronous event occur on the *gpio*. The value of **arg[0]** is expected to be an event mask of flags, constructed by computing the bit-wise OR of the desired flag values. Currently, only the following flag is supported:

ST_ARQ2 Signal on assertion of Attention Request (ARQ) triggered by ATTN line.

When the interrupt mask is enabled and ATTN line is asserted, the process is sent SIGEMT; therefore, the user should set up a handler to trap this signal (see *signal*(2)). The reason for interrupt can be obtained via the IO_STATUS request GPIO_SIGNAL_MASK.

To receive multiple interrupts, the interrupt mask must be re-enabled after each SIGEMT signal.

Refer to GPIO_STS_LINES for an explanation of how GPIO_SIGNAL_MASK affects the number of status lines on the HP 27114B interface.

Note that for the HP 28651A interface, the user can enable or disable abortion of the current transaction by asserting the interrupt line (see the ABORT_ON_EXT_INT flag of the GPIO_SET_CONFIG command).

GPIO_LOCK Lock or unlock the *gpio* interface. Setting **arg[0]** to LOCK_INTERFACE gives the calling process exclusive access to the card. The lock is incremental, meaning that if the interface is already locked by the current process, additional lock requests increment a per-open lock count maintained in the driver.

An **arg[0]** of UNLOCK_INTERFACE decrements the per-open lock count. When the total interface lock count drops to zero, the lock is cleared. The lock also can be cleared by setting **arg[0]** to CLEAR_ALL_LOCKS, which removes all locks held by the current process.

After a successful lock or unlock, **arg[1]** contains the current lock count for this open, and **arg[2]** contains the total lock count on this interface.

While the interface is locked, other processes attempting to access the interface are blocked until either the interface becomes unlocked or until the existing timeout expires (timer is started by the driver based on the existing timeout value). However, if the O_NDELAY file status flag is set (see *fcntl*(5)), the user request returns immediately with an error.

GPIO_RESET Reset the *gpio* interface. This request restores the interface to a known state; **arg[0]** has the following value:

HW_CLR For the HP 27114A and HP 27114B, set the *gpio* card to its default configuration. This command alters the control lines, the transfer counter, EDGE_LOGIC_SENSE, the handshake mode, the data path, and the PEND option (refer to section titled *Default Configuration* later in this entry).

For the HP 28651A, the *gpio* card is set to its default configuration then reconfigured to match what the configuration of the card was prior to the GPIO_RESET.

GPIO_SET_CONFIG

Set up additional parameters for the *gpio* interface as specified in **arg[0]** and **arg[1]**.

The configuration mask in **arg[0]** is constructed by computing the bit-wise OR of the flags from the list below. Since each request overwrites the previous parameter setting, it is advisable to get the current settings through an IO_STATUS *ioctl* GPIO_GET_CONFIG request.

LOGIC_SENSE_SIZE
For the HP 28651A interface only. This flag is used to mask out the "logic sense" flags described below. This is useful for determining which logic sense is enabled.

PFLG_LOGIC_SENSE
For the HP 28651A only. Define the logic sense for PFLG handshake line as "Ready when PFLG High." Default: "Ready when PFLG Low."

PCTL_LOGIC_SENSE
For the HP 28651A only. Define the logic sense for PCTL handshake line as "Control Set when PCTL High." Default: "Control Set when PCTL Low."

PDDR_LOGIC_SENSE
For the HP 28651A interface only. Define the logic sense for PDDR

handshake line as "Out direction when PDDR High." Default: "Out direction when PDDR Low."

EDGE_LOGIC_SENSE
Defines which edge of the PFLG input is used by the HP 27114A and HP 27114B cards to handshake data.

For the HP 27114A: if EDGE_LOGIC_SENSE is asserted, the busy-to-ready (falling) edge of PFLG will trigger data movement. Otherwise, the ready-to-busy edge of PFLG will trigger data movement. Default: the ready-to-busy edge of PFLG triggers data movement.

For the HP 27114B: if EDGE_LOGIC_SENSE is asserted the busy-to-ready edge of PFLG will trigger data movement while in the FIFO handshake mode; the low level of PFLG will trigger data movement if in either of the FULL handshake modes. If EDGE_LOGIC_SENSE is not asserted in the configuration mask, the ready-to-busy edge of PFLG trigers data movement if in the FIFO handshake mode; the high level of PFLG will trigger data movement if in either of the FULL handshake modes. Default: the ready-to-busy edge of PFLG triggers data movement.

The HP 28651A does not support this option.

The following handshake modes are available for the driver and the peripheral:

HANDSHAKE_MODE_SIZE
For the HP 28651A only. This flag masks out the handshake mode bits. This is useful for determining which handshake mode is enabled.

FULL_HANDSHAKE_MODE
For the HP 28651A only. This configuration mask selects the Full handshake mode, which allows peripherals to indicate a state of readiness. This is the default mode.

PULSE_HANDSHAKE_MODE
For the HP 28651A only. This configuration mask selects the Pulse handshake mode.

STROBE_HANDSHAKE_MODE
For the HP 28651A only. This configuration mask selects the Strobe handshake mode.

SLAVE_HANDSHAKE_MODE
For the HP 28651A only. This configuration mask selects the Slave handshake mode.

FIFO_MASTER For the HP 27114B only. This flag selects the FIFO_MASTER' handshake, also known as a "pulsed" handshake. This is the same handshake used by the HP 27114A. Default: FIFO_MASTER is asserted.

FULL_MASTER For the HP 27114B only. This flag selects the FULL_MASTER handshake. This handshake is useful when using *gpio*-to-*gpio* links.

FULL_SLAVE For the HP 27114B. This flag selects the FULL_SLAVE handshake. This handshake is useful when using *gpio*-to-*gpio* links.

The HP 27114A has only one handshake, the FIFO_MASTER handshake.

The following flags define the settings available for the clock source bits:

DIN_CLK_SIZE For the HP 28651A only. This flag is used to mask out the clock source bits used for latching the input data.

DIN_CLK_PFLG_BUSY_TO_READY
For the HP 28651A only. This configuration mask selects the PFLG busy-to-ready transition for latching the input data. This is the default mode.

DIN_CLK_PFLG_READY_TO_BUSY
For the HP 28651A only. This configuration mask selects the PFLG ready-to-busy transition for latching the input data.

DIN_CLK_PCTL_SET_TO_CLEAR
For the HP 28651A only. This configuration mask selects the PCTL set-to-clear transition for latching the input data.

DIN_CLK_PCTL_CLEAR_TO_SET
For the HP 28651A only. This configuration mask selects the PCTL clear-to-set transition for latching the input data.

DIN_CLK_READ_OF_LO_BYTE
For the HP 28651A only. This configuration mask latches the input data whenever the low byte of the input register is read.

DIN_CLK_READ_OF_HI_BYTE
For the HP 28651A only. This configuration mask latches the input data whenever the high byte of the input register is read.

TRNSFR_CTR_EN
For the HP 27114B only. If the configuration mask asserts the TRNSFR_CTR_EN flag, the HP 27114B will stop handshaking when the transfer counter becomes zero. This is unlike the HP 27114A which, during input transfers, continues to handshake data onto the card (up to another 66 words of data) if the peripheral supplies the data. Note: if the transfer counter is not enabled during writes where PEND is asserted, the number of bytes transfered is unknown. Default: the transfer counter is disabled.

PDIR_OPT_EN For the HP 27114B only. Whenever the PDIR_OPT_EN flag is asserted, the CTL5 line reflects the DIR output and the CTL4 line reflects the HEND output. If the PDIR_OPT_EN flag is not asserted the CTL5 line reflects the sixth control bit, CTL5, and the CTL4 line reflects the fifth control bit, CTL4. The default is PDIR_OPT_EN asserted.

Refer to GPIO_CTL_LINES for an explanation of how PDIR_OPT_EN affects the number of control lines of the HP 27114B.

PEND_OPT_EN For the HP 27114B only. When the PEND_OPT flag is asserted, the assertion of the PEND input at the frontplane will terminate the data transfer on the backplane. Default: PEND_OPT_EN is not asserted.

Refer to GPIO_STS_LINES for an explanation of how PEND_OPT_EN affects the number of status lines on the HP 27114B.

ABORT_ON_EXT_INT
For the HP 28651A only. This configuration mask causes the driver to abort the current request on external interrupts. Default: request not aborted.

The value of **arg[1]** is valid for the HP 28651A only. The parameter in **arg[1]** sets a delay to allow data to settle. The delay set in this parameter postpones assertion of PCTL or reception of PFLG by the specified nano-seconds(nsec). The range for the delay is 125-2000 nsec and the default value is 2000 nsec.

The HP 27114B allows a PCTL/PFLG delay to be set via hardware jumpers (refer to the HP 27114B Hardware Reference Manual).

GPIO_CTL_LINES
This control function allows the user to set or clear the *gpio* control lines. The **arg[0]** is an integer mask mapped onto the control lines, with the least significant bit corresponding to control line 0 (CTL0); every set bit asserts its associated control line.

Three control lines are available to the user on the HP 27114A: CTL0-CTL2.

Up to six control lines are available on the HP 27114B. Four of these control lines, CTL0-CTL3, are always available. The fifth and sixth control lines, CTL4 and CTL5, are

multiplexed at the frontplane to reflect either the outputs from the HEND and DIR bits, or the outputs from the CTL4 and CTL5 bits. The multiplexing is defined by the assertion/deassertion of the PDIR_OPT_EN flag in **arg[0]** of GPIO_SET_CONFIG.

The default state of the CTL5 line reflects the DIR output. The default state of the CTL4 line reflects the HEND output. Whenever the PDIR_OPT_EN flag is asserted this is the frontplane configuration for the CTL5 and CTL4 line outputs.

To enable the sixth control line to reflect the sixth control bit, CTL5, and the fifth control line, CTL4, to reflect the fifth control bit, the PDIR_OPT_EN flag must be deasserted. See GPIO_SET_CONFIG.

There are five control lines on the HP 28651A.

Status Requests

These requests are used to obtain information about the state of a device or the *gpio* in general. They use a calling sequence similar to that of control requests:

```
struct io_ctl_status {
  int type;
  int arg[3];
} gpio_status;

ioctl (fildes, IO_STATUS, &gpio_status);
```

The **type** member specifies the type of information to return in the **arg[ ]** array. The following values for **type** are supported by *gpio*:

GPIO_TIMEOUT
: Return the interface's timeout in microseconds in **arg[0]**. Zero is returned when the timeout is infinite.

GPIO_WIDTH
: Return the interface's path width in bits in **arg[0]**.

GPIO_SIGNAL_MASK
: Return the reason for the last interrupt in **arg[0]**. The mask returned has bits set to indicate the reason(s) for the last SIGEMT. Bit definitions are the same as the corresponding IO_CONTROL request bits.

GPIO_LOCK
: If the device is locked to a process, return that process ID in **arg[0]** and the interface lock count in **arg[1]**. If the device is not locked, **arg[0]** contains **-1**.

GPIO_GET_CONFIG
: Return to the user the configuration parameters specified in the GPIO_SET_CONFIG for IO_CONTROL. The value of **arg[0]** is the configuration mask described above and **arg[1]** is the handshake delay value in nano-seconds.

GPIO_STS_LINES
: This status function gives the user access to the *gpio* status lines. The value of **arg[0]** is an integer mask similar to that of GPIO_CTL_LINES, with the state of status line 0 (*STS0*) being returned in the least-significant bit. GPIO_STS_LINES returns only the state of those lines; they cannot be set programmatically.

  Two status lines are available to the user on the HP 27114A.

  Up to six status lines are available on the HP 27114B. Four of these status lines, STS0-STS3, are always available. The fifth status line, STS4, is multiplexed between the PEND circuit and the STS4 bit. The default state for the fifth status line sends input to the PEND circuit, eventhough the PEND option is disabled. To enable the fifth status line for input to the STS4 bit, the PEND option must be disabled (see control request GPIO_SIGNAL_MASK). The sixth status line, STS5, is multiplexed between the attention interrupt circuit, ATTN, and the STS5 bit. The default state for the sixth status line sends input to the ATTN circuit, eventhough the card is not enabled to generate interrupts. To enable the sixth status line for input to the STS5 bit, interrupts (ARQs) must be disabled (see control request GPIO_SIGNAL_MASK).

Five status lines are available to the user on the HP 28651A.

GPIO_INTERFACE_TYPE
Return the type of interface in **arg[0]**. This is a 32-bit value as follows:

For the HP 27114A card: (left to right)

bits 0-15	0
bits 16-19	0
bit 20	parity
bits 21-23	card revision number
bits 24-31	card ID

For the HP 27114B card: (left to right)

bits 0-15	0
bits 16-23	card revision number
bits 24-31	card ID

For the HP 28651A card: (left to right)

bits 0-15	GPIO1_INTERFACE
bits 16-19	card revision number
bits 20-31	card ID

The HP 27114A card will have a revision number of 0 or 1.
The HP 27114B card will have a revision number of 2 or greater.

Extended Status Request
To obtain several status variables in one request, the following request can be made:

```
struct io_environment gpio_env;
ioctl(fildes, IO_ENVIRONMENT, &gpio_env);
```

where the **io_environment** structure includes the following members:

```
int interface_type;
int timeout;
int status;
int signal_mask;
int width;
int locking_pid;
unsigned int config_mask;
unsigned short delay;
```

Default Configuration
The default configuration of any *gpio* interface is:

Timeout	one hour (infinite on the HP 28651A)
Path Width	16 bits (8 bits on the HP 28651A)
Interrupts	disabled
Locking	unlocked
User control lines	all zero (set on the HP 28651A)

Additional defaults to the HP 27114B interface are:

PEND	Disabled
Transfer Counter	Disabled
Edge_logic_sense	not asserted; the ready-to-busy edge of PFLG triggers data movement. This is the same for the HP 27114A.
Handshake mode	FIFO_MASTER
Data path	empty; the value of the data lines is unknown

ERRORS
A **-1** return value for a driver request indicates that an error occurred; **errno** is set to indicate the reason. In addition to errors defined in *open*(2), *close*(2), *read*(2), *write*(2), and *ioctl*(2), a driver request can fail if any of the following is true:

[EACCES] The access to the specific device file cannot be granted without the proper minor number.

[EACCES] The interface is currently locked via GPIO_LOCK.

[EFAULT] I/O request specified odd byte count or odd address on 16-bits GPIO data-width mode.

[EINTR] An interface power failure occurred during the processing of this request; the device might have lost state.

[EINVAL] An attempt was made to unlock an interface that was not locked.

[EINVAL] Invalid command or parameter.

[EIO] Some unclassified error occurred.

[EMFILE] The maximum number of simultaneous opens on this interface exceeded.

[ENXIO] There is no bus interface associated with the device file.

[EPERM] An attempt is made to unlock when lock is not owned by this user.

[ERANGE] The interface lock count limit exceeded.

[ETIMEDOUT] The transaction did not complete within the timeout specified.

WARNINGS

Processes that use GPIO_LOCK should clear all locks before exiting. The driver attempts to clear them if the process terminates unexpectedly; however, a lock might be left outstanding if the locker dies after creating new file descriptors (via *fork*(2) or *dup*(2)) that refer to the same device file. Ensuring that all open file descriptors on that interface are closed remedies the situation.

DEPENDENCIES

Series 800

Note that this interface applies to both the HP 27114A/B (CIO AFIs) and HP 28651A (HP-PB) cards which have very dissimilar hardware features. *gpio* provides as consistent an interface as possible across all these cards, but there are still significant differences. Because of this, users of this interface should consult the appropriate hardware documentation and read carefully the hardware-specific information given in this manual entry in the following *ioctl* command definitions:

GPIO_RESET
GPIO_SIGNAL_MASK
GPIO_SET_CONFIG
GPIO_CTL_LINES
GPIO_STS_LINES
GPIO_GET_CONFIG
GPIO_INTERFACE_TYPE
Path Width under Default Configuration

Links from HP 27114A to HP 27114A are not supported. Links from HP 27114A to HP 28651A are supported only when the HP 28651A is configured as slave. Links from the HP 27114A to the HP 27114B are supported only when the HP 27114B is configured as slave.

AUTHOR

gpio was developed by HP.

FILES

/dev/gpio*

SEE ALSO

ioctl(2), signal(2), particular device documentation.

NAME

CRT graphics - information for CRT graphics devices

Remarks:

This information is valid for Series 300/400 only.

DESCRIPTION

CRT graphics devices are frame-buffer based raster displays. These devices use memory-mapped I/O to obtain much higher performance than is possible with tty-based graphics terminals. CRT graphics devices can be accessed directly using this interface, although access through the STARBASE libraries is recommended (see *starbase*(3G)). Direct access to frame-buffer devices entails precise knowledge of the frame-buffer architecture being used. Input cannot be piped into or redirected to frame-buffer devices because they are not serial devices.

Special (device) files for CRT graphics devices are character special files with major number **12**.

The minor number for CRT graphics devices is of the form:

0xSSTTXX

where SS is a one-byte select code number, TT is a one-byte type specifier, and XX is zero or contains device-specific information as defined in the appropriate Starbase Device Drivers Library.

The type field in the minor number is defined as follows:

0 Auto-configures to one of the following:

- Low-resolution graphics device at physical address 0x520000 (if present).
- High-resolution graphics device at physical address 0x560000 if low resolution device at 0x520000 not present.

1 High-resolution graphics device at physical address 0x560000 (unless there is no low resolution device at 0x520000, in which case type 1 is invalid).

2 High- or low-resolution graphics device at the select code specified by the select code field in the minor number.

Communication with a CRT graphics device is begun with an *open* system call. Multiple processes may concurrently have the graphics device open. A graphics device can be accessed by multiple processes at once; however, each process overwrites the output of the others unless one of the locking mechanisms described below, or some other synchronization mechanism, is used. The locking mechanisms described here are intended for cooperating processes only (see the description of the GCLOCK *ioctl* call below).

The *close* system call shuts down the file descriptor associated with the graphics device.

The *read* and *write* system calls are undefined and always return an error.
For either case, **errno** is set to ENODEV.

The *ioctl* system call is used to control the graphics device. For all frame buffers, the data bytes scan from left to right and from top to bottom. A pixel, which is a visible dot on the screen, is associated with a location in the frame buffer. Some devices map individual bits to pixels; some map bytes or parts of bytes to pixels (see the GCDESCRIBE *ioctl* request).

The following are valid *ioctl* requests:

GCDESCRIBE Describe the size, characteristics and mapped regions of the frame-buffer. The information is returned to the calling process in a **crt_frame_buffer_t** data structure. The parameter is defined as **crt_frame_buffer_t** **arg*;. The **crt_frame_buffer_t** data structure is described in the file **<sys/graphics.h>**. Although some structure fields contain addresses of one or more frame-buffer device regions, the values of these fields are not always defined. Only after a successful GCMAP command is issued (see below) are the correct addresses returned so the user can access the frame-buffer regions directly using the returned addresses.

GCID Provide a device identification number. The parameter is defined as **int** **arg*;. The information returned from this request is a subset of the information provided by GCDESCRIBE, and is provided here for backward compatibility only. The device identification numbers are listed in the file **<sys/graphics.h>**

GCON, GCOFF Turn graphics on or off. These operations are valid for devices whose CRT_GRAPHICS_ON_OFF bit is set in the **crt_attributes** field of the **crt_frame_buffer_t** data structure returned by the GCDESCRIBE command. Otherwise, these commands have no effect. The parameter *arg* should be set to **0**.

GCAON, GCAOFF Turn alpha on or off. These operations are valid for devices whose CRT_ALPHA_ON_OFF bit is set in the **crt_attributes** field of the **crt_frame_buffer_t** data structure returned by the GCDESCRIBE command. Otherwise, these commands have no effect. Parameter *arg* should be set to **0**.

GCMAP Map the CRT graphics device into the user address space at the address specified in the ioctl argument. The parameter is **char** ***arg**;. The value ***arg** is used as a requested address. The actual mapping address is then returned in ***arg**. If ***arg** is set to 0 before the call, the system selects the first available address. Only processes that make this request can directly access the frame-buffer memory and control registers. After a successful GCMAP call, the fields **crt_frame_base** and **crt_control_base** in the **crt_frame_buffer_t** data structure (returned by a subsequent GCDESCRIBE *ioctl* call), hold the valid addresses of these two regions of the frame-buffer.

GCUNMAP Remove the mapping of the CRT graphics device from the user address space. The parameter is **char** ***arg**;. The value ***arg** is set to the actual mapping address returned as ***arg** by the GCMAP call that originally mapped the device into the user address space.

GCLOCK Provide for exclusive use of the graphics device by cooperating processes. The calling process either locks the device and continues or is blocked. Blocking in this case means that the call returns only when the frame-buffer is available or when the call is interrupted by a signal. Waiting occurs if another process has previously locked this frame-buffer using the GCLOCK command and has not yet executed a GCUNLOCK command. The GCLOCK command does not prevent other non-cooperating processes from writing to the frame-buffer; thus, GCLOCK is an advisory lock only. The parameter *arg* should be set to **0**. Any attempt to lock the device more than once by the same process fails, and causes **errno** to be set to EBUSY.

Once the display is acquired for exclusive use (and thus locked), all signals sent to the process that otherwise would have been caught by the process *at the time of the* GCLOCK *call*, are withheld (blocked) until GCUNLOCK is requested. Any attempt to modify the signal mask of the process (see *sigsetmask*(2)) before a GCUNLOCK request is made will not have any effect on these blocked signals. The signals are not blocked until the lock is actually acquired and might be received while still awaiting the lock.

The signal SIGTSTP is blocked whether or not it is currently being caught. The signals SIGTTIN and SIGTTOU are also blocked on frame-buffer devices where the ITE does not output to the device while it is locked.

Except for the three signals mentioned above, this call does not block signals that the process did not expect to catch, nor does it block signals that cannot be caught or ignored.

GCLOCK_MINIMUM Provide for exclusive use of the graphics device by cooperating processes. This request has the same effect on the graphics device as does the GCLOCK request. However, this call does not block any signals as does the GCLOCK request. The GCLOCK_MINIMUM command does not prevent other non-cooperating processes from writing to the frame-buffer; thus, GCLOCK_MINIMUM is an advisory lock only. The parameter *arg* should be set to **0**. Any attempt to lock the device more than once by the same process fails, and causes **errno** to be set to EBUSY.

GCUNLOCK Relinquish exclusive use of the CRT graphics device. The parameter *arg* should be set to **0**. Any attempt to unlock a graphics device which is locked by a different process will fail and cause **errno** to be set to EPERM.

GCUNLOCK_MINIMUM
Relinquish exclusive use of the CRT graphics device. The parameter *arg* should be set

to **0**. Any attempt to unlock a graphics device which is locked by a different process will fail and cause **errno** to be set to EPERM.

GCSLOT — Provide pertinent information about the calling process's participation in the system-wide graphics locking mechanism (see the discussion under GCLOCK above). The GCSLOT request does *not* carry out any actual locking functionality. The lock information is returned to the calling process in a **gcslot_info** data structure. The parameter is defined as **gcslot_info** **arg*;. The **gcslot_info** data structure is defined in the file **<sys/graphics.h>.**

GCSTATIC_MAP — Prevent the Internal Terminal Emulator (ITE) from modifying the device's color map.

GCVARIABLE_MAP — Allow the Internal Terminal Emulator (ITE) to modify the device's color map.

One shared memory descriptor (see *shmget*(2)) is assigned to each graphics device by the GCMAP request. The GCSLOT request attaches a separate shared memory object that consumes a second shared memory descriptor. Each shared memory descriptor is accessible only through its graphics interface. Thus, any attempt to access them through *shmat*(2)), *shmctl*(2)), *shmdt*(2)), etc. results in EACCES errors.

ERRORS

[ENODEV] Attempted to use *read* or *write* system calls on the device.

[EINVAL] An invalid *ioctl* command was made.

[EBUSY] Attempt to lock a device which is already locked by the same process.

[EPERM] Attempt to unlock a device which is locked by a different process.

[ENXIO] No such device or too many opens.

[ENOSPC] Cannot allocate required resources for mapping.

[ENOMEM] Cannot allocate sufficient memory for mapping.

[EACCES] Illegal attempt to access shared memory descriptor.

[EMFILE] Cannot allocate required resources for locking mechanism.

[ENOTTY] Bad ioctl command.

SEE ALSO

starbase(3G), mknod(1M), open(2), close(2), ioctl(2), sigsetmask(2). mknod(1M).

NAME

hil - HP-HIL device driver

SYNOPSIS

#include <sys/hilioctl.h>

DESCRIPTION

HP-HIL, the Hewlett-Packard Human Interface Link, is the Hewlett-Packard standard for interfacing a personal computer, terminal, or workstation to its input devices. *hil* supports devices such as keyboards, mice, control knobs, ID modules, button boxes, digitizers, quadrature devices, bar code readers, and touchscreens.

On systems with a single link, HP-HIL device file names use the following format:

/dev/hil*n*

where *n* represents a single digit that specifies the physical HP-HIL device address, which ranges from 1 to 7. For example, **/dev/hil3** is used to access the third HP-HIL device.

On systems with more than one link, HP-HIL device file names use the following format:

/dev/hil_*m.n*

where *m* represents the link logical unit number, and *n* represents the physical HP-HIL device address. For example, **/dev/hil_0.2** would be used to access the second device on link logical unit zero. Likewise, **/dev/hil_12.7** references the seventh device on link logical unit number twelve.

Note that HP-HIL device addresses are determined only by the order in which devices are attached to the link. The first device attached to the link becomes device one, the second device attached becomes device two, etc.

HP-HIL devices are classified as "slow" devices. This means that system calls to *hil* can be interrupted by caught signals (see *signal*(5)).

hil can only read HP-HIL keyboards in raw keycode mode. Raw keycode mode means that all keyboard input is read unfiltered. HP-HIL keyboards return keycodes that represent key press and key release events.

Use *hilkbd*(7) to read mapped keycodes from HP-HIL keyboards. Use the Internal Terminal Emulator (ITE) described in *termio*(7) to read ASCII characters from HP-HIL keyboards.

System Calls

open(2) gives exclusive access to the specified HP-HIL device. Any previously queued input from the device is discarded. If the device is a keyboard, it is opened in raw keycode mode. A side effect of opening a keyboard in raw keycode mode is that the ITE (see *termio*(7)) and mapped keyboard driver (see *hilkbd*(7)) lose input from that keyboard until it is closed. Only device implemented auto-repeat functionality is available while in raw keycode mode (see HILER1 and HILER2).

The file status flag, O_NDELAY, can be set to enable non-blocking reads (see *open*(2)).

close(2) returns an HP-HIL keyboard to mapped keycode mode, making its input available to the ITE or mapped keyboard driver (see *hilkbd*(7)).

read(2) returns data from the specified HP-HIL device, in time-stamped packets:

```
unsigned char packet_length;
unsigned char time_stamp[4];
unsigned char poll_record_header;
unsigned char data[ packet_length - 6 ];
```

packet_length specifies the number of bytes in the packet including itself, and can range from six to twenty bytes. *time_stamp*, when re-packed into an integer, specifies the time, in tens of milliseconds, that the system has been running since the last system boot. The most significant byte of the time stamp is *time_stamp*[0]. *poll_record_header* indicates the type and quantity of information to follow, and reports simple device status information. The number of data bytes is device dependent. Refer to the text listed in SEE ALSO for descriptions of the *poll_record_header* and device-specific data.

Usually two system calls are required to read each data packet, the first system call reads the data packet length; the second system call reads the actual data packet. Some devices always return the same amount of data in each packet, in which case the count and the packet can both be read in the same system call.

If the file status flag, O_NDELAY, is set and no data is available, *read*(2) returns 0 instead of blocking.

write(2) is not supported by *hil*.

select(2) can be used to poll for available input from HP-HIL devices. *select*(2) for write or for exception conditions always returns a false indication in the file descriptor bit masks.

ioctl(2) is used to perform special operations on HP-HIL devices. *ioctl*(2) system calls all have the form:

int ioctl(int fildes, int request, char *arg);

The following *request* codes are defined in **<sys/hilioctl.h>**:

HILID Identify and Describe

This request returns the Identify and Describe Record in the **char** variable to which *arg* points, as supplied by the specified HP-HIL device. The Identify and Describe Record is used to determine the type and characteristics of each device connected to the link. The Identify and Describe Record can vary in length from 2 to 11 bytes. The record contains at least:

- A Device ID byte, and
- A Describe Record Header byte.

The Device ID byte is used to identify the general class of a device, and its nationality in the case of a keyboard or keypad. The Describe Record Header byte describes the position report capabilities of the device. The Describe Record Header byte also indicates if an I/O Descriptor byte follows at the end of the Describe Record. It also indicates support of the Extended Describe and the Report Security Code requests. If the device is capable of reporting any coordinates, the Describe Record contains the device resolution immediately after the Describe Record Header byte. If the device reports absolute coordinates, the maximum count for each axis is specified after the device resolution. The I/O Descriptor byte indicates how many buttons the device has. The I/O Descriptor byte also indicates device proximity detection capabilities and specifies Prompt/Acknowledge functions. All HP-HIL devices support the Identify and Describe request.

HILPST Perform Self Test

This request causes the addressed device to perform its self test, and returns the one-byte test result in the **char** variable to which *arg* points. A test result of zero indicates a successful test, non-zero results indicate device-specific failures. All HP-HIL devices support the Self Test request.

HILRR Read Register

The Read Register request expects an HP-HIL device register address in the **char** variable to which *arg* points, and returns the one-byte contents of that register in **arg*. The Extended Describe Record indicates whether a device supports the Read Register request.

HILWR Write Register

The Write Register request expects **arg* to contain a record containing one or more packets of data, each containing the HP-HIL device register address and one or more data bytes to be written to that register. There are two types of Register Writes. Type 1 can be used to write a single byte to each individual device register. Type 2 can be used to write several bytes to one register. The Extended Describe Record indicates if a device supports either or both types of register write requests.

HILRN Report Name

The Report Name request returns the device description string in the character array to which *arg* points. The string may be up to fifteen characters long. The Extended Describe Record indicates support of the Report Name request.

HILRS Report Status

The Report Status request returns the device-specific status information string in the character array to which *arg* points. The string can be up to fifteen bytes long. The Extended

Describe record indicates support of the Report Status request.

HILED Extended Describe

The Extended Describe request returns the Extended Describe Record in the character array to which *arg* points. The Extended Describe Record may contain up to fifteen bytes of additional device information. The first byte is the Extended Describe Header, which indicates whether a device supports the Report Status, Report Name, Read Register, or Write Register requests. If the device implements the Read Register request, the maximum readable register is specified. If the device supports the Write Register request, the Extended Describe Record specifies whether the device implements either or both of the two types of register writes and the maximum writeable register. If the device supports Type 2 register writes, the maximum write buffer size is specified. The Extended Describe Record can also contain the localization (language) code for a device. Support of the Extended Describe request is indicated in the Describe Record Header byte.

HILSC Report Security Code

The Report Security Code request returns the Security Code Record in the character array to which *arg* points. The Security Code Record can be between one and fifteen bytes of data that uniquely identifies that particular device. Applications can use this request to implement a hardware "key" that restricts each copy of the application to a single machine or user. An application can read the Security Code Record from an HP-HIL ID Module and then verify that the application is running on a specific machine or that the application is being used by a legitimate user. Devices indicate support of the Report Security Code request in the Describe Record Header.

HILER1 Enable Auto Repeat Rate = 1/30 Second

This request is used to enable the "repeating keys" feature implemented by the firmware of some HP-HIL keyboard and keypad devices. It also sets the cursor key repeat rate to 1/30 sec. This request does not use *arg*.

HILER2 Enable Auto Repeat Rate = 1/60 Second

This request is used to enable the "repeating keys" feature implemented in the firmware of some HP-HIL keyboard and keypad devices. It also sets the cursor key repeat rate to 1/60 sec. This request does not use *arg*.

HILDKR Disable Keyswitch Auto Repeat

This request turns off the "repeating keys" feature implemented in the firmware of some HP-HIL keyboard and keypad devices. This request does not use *arg*.

HILP1..HILP7 Prompt 1 through Prompt 7

These seven requests are supported by some HP-HIL devices to give an audio or visual response to the user, perhaps indicating that the system is ready for some type of input. A device specifies acceptance of these requests in the I/O Descriptor Byte in the Describe Record. These requests do not use *arg*.

HILP Prompt (General Purpose)

This request is intended as a general purpose stimulus to the user. Devices accepting this request indicate so in the I/O Descriptor Byte in the Describe Record. This request does not use *arg*.

HILA1..HILA7 Acknowledge 1 through Acknowledge 7

These seven requests are intended to provide an audio or visual response to the user, generally to acknowledge a user's input. The I/O Descriptor Byte in the Describe Record indicates whether an HP-HIL device implements this request. These requests do not use *arg*.

HILA Acknowledge (General Purpose)

The Acknowledge request is intended to provide an audio or visual response to the user. Devices accepting this request indicate so in the I/O Descriptor Byte in the Describe Record. This request does not use *arg*.

ERRORS

[EBUSY]	The specified HP-HIL device is already opened.
[EFAULT]	A bad address was detected while attempting to use an argument to a system call.
[EINTR]	A signal interrupted an *open*(2), *read*(2), or *ioctl*(2) system call.
[EINVAL]	An invalid parameter was detected by *ioctl*(2).
[ENXIO]	No device is present at the specified address; see WARNINGS, below.
[EIO]	A hardware or software error occurred while executing an *ioctl*(2) system call.
[ENODEV]	*write*(2) is not implemented for HP-HIL devices.

WARNINGS

An ENXIO error is returned by *open*(2) and *ioctl*(2) if any attempt is made to access a device while *hil* is reconfiguring the link during power-failure recovery.

hil cannot detect whether or not a device executed an *ioctl*(2) request.

HP-HIL devices have no status bit available to indicate whether they support the HILER1, HILER2, or HILDKR requests.

AUTHOR

hil was developed by the Hewlett Packard Company.

FILES

/dev/hil[1-7]
/dev/hil_*.[1-7]

SEE ALSO

close(2), errno(2), fcntl(2), ioctl(2), open(2), read(2), select(2), signal(5), hilkbd(7), termio(7).

For detailed information about HP-HIL hardware and software in general, see the *HP-HIL Technical Reference Manual*.

NAME

hilkbd - HP-HIL mapped keyboard driver

DESCRIPTION

HP-HIL, the Hewlett-Packard Human Interface Link, is the Hewlett-Packard standard for interfacing a personal computer, terminal, or workstation to its input devices. *hilkbd* supplies input from all mapped keyboards on a specified HP-HIL link.

hilkbd returns mapped keycodes, not ASCII characters. "Raw" keycodes are the individual key downstrokes and upstrokes, and are different for each type of keyboard. *hilkbd* maps the raw input into the keycodes and protocol expected by the HP-UX, Pascal Workstation, and BASIC/UX operating systems. The *hil*(7) driver can usurp a keyboard from *hilkbd* by changing it from mapped mode to raw mode.

System Calls

open(2) gives exclusive access to the keyboard. If there is an ITE (internal terminal emulator) associated with the keyboard, the ITE loses input from the keyboard until the keyboard device is closed. Any previous queued input for the keyboard device is flushed from the input queue.

close(2) returns control of the keyboard to the ITE, if present. Any unread input is discarded at that time.

read(2) returns data from the keyboard in time-stamped packets:

```
unsigned char time_stamp[4];
unsigned char status;
unsigned char data;
```

time_stamp, when repacked into an integer data type of four or more bytes, specifies the time since an arbitrary point in the past (e.g., system start-up time). This point does not change between packets, but time during a power failure may or may not be counted. The time is in units of tens of milliseconds.

The *status* byte encodes the state of the keyboard **Shift** and **Ctrl** keys:

0x8X	shift and control
0x9X	control only
0xAX	shift only
0xBX	no shift or control

The *data* byte contains the actual keystroke.

If the file status flag O_NDELAY is set, *read*(2) returns **0** instead of blocking, when no data is available. The *read*(2) system call on an HP-HIL keyboard is considered "slow"; that is, it can be interrupted by caught signals (see *signal*(2)).

write(2) is not supported by *hilkbd.*

select(2) can be used to poll for input to read from *hilkbd* devices. *select*(2) for write or for exceptional conditions always returns a false indication in the bit masks.

ioctl(2) is used to perform special operations on the device. *ioctl*(2) system calls have the form:

```
int ioctl(int fildes, int request, char *arg);
```

The following *hilkbd* request codes are defined in **<sys/hilioctl.h>**:

KBD_READ_CONFIG
: Read the configuration code.

  This request returns a one-byte configuration code in the **char** variable to which *arg* points. This contains a field, defined by KBD_IDCODE_MASK, which specifies the keyboard identification code. The possible values of this field are defined in the header file, and this identification code affects interpretation of the language code. All other fields in the configuration code are currently undefined.

KBD_READ_LANGUAGE
: Read the language code.

This request returns a one-byte language code, as read from the keyboard, in the **char** variable to which *arg* points. If there is more than one keyboard, the language is taken from the first keyboard on the link. Interpretation of the language code is affected by the keyboard identification field within the configuration code.

KBD_STATUS Read the keyboard status register.

This request returns a one-byte value containing bit flags specifying the state of the shift and control keys in the **char** variable to which *arg* points:

KBD_STAT_LEFTSHIFT	The left shift key is up
KBD_STAT_RIGHTSHIFT	The right shift key is up
KBD_STAT_SHIFT	Both shift keys are up
KBD_STAT_CTRL	The control key is up

Other bits are undefined.

KBD_REPEAT_RATE
Set the keyboard auto-repeat rate.

The one-byte value to which *arg* points is the negative of the repeat period, in tens of milliseconds. The repeat rate is the reciprocal of the repeat period. A parameter of zero disables auto-repeat.

KBD_REPEAT_DELAY
Set the keyboard auto-repeat delay.

The one-byte value to which *arg* points is the negative of the repeat delay, in tens of milliseconds.

KBD_BEEP Cause an audible beep.

The one-byte value to which *arg* points specifies the volume of the beep, within the range 0 through KBD_MAXVOLUME. Implementations with fewer than KBD_MAXVOLUME discrete levels of volume will scale the parameter into the smaller range.

ERRORS

[EINVAL] An invalid parameter was detected by *ioctl*(2).

[EINTR] A signal was caught during a *read*(2) system call.

[ENXIO] No keyboard is present on the HP-HIL link specified by the minor number.

[ENODEV] An attempt was made to use *write*(2) using *hilkbd*.

[EBUSY] The device is already open.

AUTHOR

hilkbd was developed by the Hewlett-Packard Company.

FILES

/dev/hilkbd*

SEE ALSO

termio(7), hil(7), mknod(1M), select(2), signal(2).

NAME

hpib - Hewlett-Packard Interface Bus driver

SYNOPSIS

```
#include <sys/hpibio.h>
```

DESCRIPTION

HP-IB is Hewlett-Packard's implementation of the Institute of Electrical and Electronic Engineers Standard Digital Interface for Programmable Instrumentation (IEEE Std 488-1978). This section describes the use of the HP-IB driver in the HP-UX system.

Auto-addressed Files vs. Raw Bus Files

A major distinction is made in the HP-UX driver between "auto-addressed" files and "raw bus" files. An auto-addressed file is associated with a specified address on the HP-IB. The user need not be concerned with any HP-IB addressing or commands; the driver handles device addressing and unaddressing during data transfers. However, the user is limited to transactions to and from a single device. A raw bus file, on the other hand, gives the user access to the entire HP-IB; responsibility for all commands and addressing lies with the user. The raw bus file is typically used to access multiple devices on the same bus, as well as provide universal device commands such as interface clear and parallel poll.

Although differences exist between auto-addressed and raw bus files, the user/driver interface is consistent across both types. Therefore, each category of requests is presented with separate subsections for auto-addressed and raw bus files.

Naming Convention

HP-IB device files are named according to the following format:

```
/dev/hpib/#[a#]
```

where the first # specifies the bus number (assigned by the administrator) and the second # specifies the address on that bus. Device files without an address suffix denote the raw bus. Files with the address suffix are auto-addressed.

Device Attributes

HP-IB attributes are classified into two groups, per-open and per-interface. File descriptors obtained from separate `open()` requests have separate per-open attributes (see *open*(2)); changing an attribute from the per-open group affects requests on that file descriptor only. Attributes in the per-interface group are shared by all file descriptors on that interface; changing an attribute from one file descriptor affects all users of the interface.

The per-open set of attributes and the driver requests to change them are listed in the following table. All other attributes are per-interface.

Attribute	Driver Request
timeout	`HPIB_TIMEOUT`
write termination mode	`HPIB_EOI`
read termination pattern	`HPIB_READ_PATTERN`
read termination reason	`HPIB_TERM_REASON`
signal mask	`HPIB_SIGNAL_MASK`
lock count	`HPIB_LOCK`
wait events	`HPIB_WAIT_ON_STATUS`

Transfer Requests

The standard `read()` and `write()` requests are used for data transfer over HP-IB (see *read*(2) and *write*(2)). However, their actions are slightly different for each type of file. Raw bus files place data directly onto the bus. No addressing or unaddressing of devices is done by the driver; this is the user's responsibility.

On the other hand, the driver does all addressing for transactions with auto-addressed files. The actual sequence of events is:

UNL, *<device addressing>*, *<data>*, *<terminator>*

All write requests end when the specified number of bytes has been transferred over the bus. Optionally, the HP-IB `END` message can be sent with the last byte written; this is controlled via the `HPIB_EOI` request. All read requests end when the specified number of bytes has been read over the bus or when the

device asserts EOI. In addition, a single character can be designated to end the read operation via the HPIB_READ_PATTERN request.

Control Requests

Control requests cause some action on the bus. All such requests have the same format:

```
struct io_ctl_status {
     int type;
     int arg[3];
} hpib_control;

ioctl(fildes, IO_CONTROL, &hpib_control);
```

In the io_ctl_status structure, the type field specifies the type of control function required, while the arg array holds any associated arguments. The defined values for type and their use are described as follows:

HPIB_TIMEOUT
: Set the timeout. If any transaction for this file takes longer than arg[0] microseconds, it is aborted with a status of ETIMEDOUT returned to the user. This is used mainly for detecting device failure. A timeout of 0 is equivalent to infinity; that is, no transaction will time out.

HPIB_WIDTH
: Set the width of the interface. This request specifies the number of valid data lines on transfers; arg[0] holds the desired interface width in bits. All future read requests inspect only the least significant arg[0] data lines, and all future writes present data on only those lines. The state of all other data lines is indeterminate.

HPIB_SPEED
: Set the transfer speed of the interface. The desired data transfer speed in kilobytes per second is specified in arg[0]. Note that this value is advisory only, and is typically used by the driver to determine the method of data transfer.

HPIB_EOI
: Enable/disable EOI assertion on writes. If arg[0] is nonzero, all subsequent writes end with EOI asserted on the last byte transferred. A zero arg[0] disables EOI assertion.

HPIB_SYSTEM_CTLR
: Make the interface system controller or non-system controller. If arg[0] is nonzero, the interface becomes the system controller. A zero in arg[0] sets the interface to non-system controller. This request is applicable to raw bus files only.

HPIB_READ_PATTERN
: Enable or disable pattern matching on reads. If arg[0] is nonzero, all subsequent reads terminate when the pattern specified in arg[1] is encountered in the input stream. This termination condition is subject to all other termination conditions in effect for the file. Only the *n* least significant bits of the pattern are used in the match, where *n* is the interface's current width, set via HPIB_WIDTH. A zero arg[0] disables read pattern matching.

HPIB_SIGNAL_MASK
: Define signaling events. This request allows the calling process to receive a signal when some event occurs on the HP-IB. The event or events are specified by computing the bitwise inclusive OR of the values from the list below, and placing the mask in arg[0]. All of these events can be enabled on raw bus files, but only ST_SRQ and ST_PPOLL apply to auto-addressed files.

  ST_SRQ — Signal on assertion of Service Request (SRQ).

  ST_PPOLL — Signal when device responds to Parallel Poll.

  ST_REN — Signal when interface enters remote state.

  ST_ACTIVE_CTLR — Signal when interface becomes active controller.

  ST_TALK — Signal when interface is addressed to talk.

  ST_LISTEN — Signal when interface is addressed to listen.

ST_IFC Signal on assertion of Interface Clear (IFC).

ST_DCL Signal on receipt of Device Clear (DCL).

ST_GET Signal on receipt of Group Execute Trigger (GET).

When any subsequent flagged event occurs, the process is sent **SIGEMT**. The user should set up a handler to trap this signal via **signal()** or **sigvector()** (see *signal*(2) or *sigvector*(2)). The reason for interrupt can be obtained via the **IO_STATUS** request **HPIB_SIGNAL_MASK**. Each request overwrites the previous mask for the file; therefore events can be disabled by using a zero **arg[0]**.

If **ST_PPOLL** is flagged, the user supplies additional information in the **arg** array. For raw bus files, the low-order bytes of **arg[1]** and **arg[2]** contain eight-bit masks with each bit corresponding to a Data I/O (DIO) line and the least significant bit mapped to DIO1. When a device responds to parallel poll, it asserts the appropriate line; **arg[1]**'s bits indicate the parallel poll sense of this assertion. Bits set in **arg[2]** indicate that the corresponding address is capable of responding to polling. For auto-addressed files, **arg[1]** specifies the parallel poll sense of the assigned device's response to parallel poll. Parallel poll interrupts can be enabled only if the interface is the active controller.

HPIB_LOCK Lock or unlock the HP-IB interface. Setting **arg[0]** to **LOCK_INTERFACE** locks the HP-IB interface, giving the calling process exclusive access to the card and bus. The lock is incremental; that is, if the interface is already locked by the current process, additional lock requests increment a per-open lock count maintained in the driver.

An **arg[0]** of **UNLOCK_INTERFACE** decrements the per-open lock count; when the total interface lock count drops to zero, the lock is cleared. The lock can also be cleared by setting **arg[0]** to **CLEAR_ALL_LOCKS**, which removes all locks held by the current process.

After a successful lock or unlock, **arg[1]** contains the current lock count for this open, and **arg[2]** contains the total lock count on this interface.

While the interface is locked, other processes that attempt to access the bus or interface are blocked until either the interface becomes unlocked or the process's per-open timeout (set via **HPIB_TIMEOUT**) expires. However, if the **O_NDELAY** file status flag is set (see *fcntl*(5)), the user request fails and returns immediately with the EACCES error. See the Summary of Privilege Requirements section for a list of user requests that might block.

HPIB_ADDRESS Set the HP-IB address to which the interface responds when it is not the active controller. The bus address is set via **arg[0]**, and must be between 0 and 30 decimal. Two additional flags, **HPIB_TALK_ALWAYS** and **HPIB_LISTEN_ALWAYS**, can be set by computing the bitwise inclusive OR of their values with the address. These flags enable the interface to talk, and/or listen always, respectively. This request is applicable to raw bus files only.

HPIB_RESET Reset the device or bus, depending on which of the following values is in **arg[0]**:

DEVICE_CLR Address the device and send a selective device clear (SDC) command. This applies only to auto-addressed files.

BUS_CLR Assert Interface Clear (IFC) and Remote Enable (REN), and clear Attention (ATN).

HW_CLR Reset bus interface card. The card is self-tested and if the card is system controller, IFC is pulsed. All other card state information is preserved. This applies to raw bus files only.

HPIB_PPOLL_RESP
Control the interface's response to parallel poll. When the interface is not acting as active controller, it can be enabled to respond to parallel polling by the current active controller. If **arg[0]** is nonzero, the interface responds to parallel poll. **arg[1]**

specifies the DIO line on which the card responds; `arg[1]` has a value between `0` and `7`, with a value of `0` mapping to DIO1, `1` mapping to DIO2, and so forth. The parallel poll sense of the response is determined by `arg[2]`. An `arg[0]` of zero disables the interface's response to parallel poll.

For auto-addressed files, the file's associated device address is configured, rather than the interface.

`HPIB_PPOLL_IST` Enable or disable response to parallel poll. If `arg[0]` is nonzero, the interface responds to parallel poll. An `arg[0]` of zero disables the interface's response. This differs from the previous request, because the parallel poll sense and address of the interface's response are unchanged. This request applies to raw bus files only.

`HPIB_REN` Place a device into or out of the remote state. For a raw bus file, this request merely sets or clears the Remote Enable line, depending on whether *arg[0]* is nonzero or zero respectively. For auto-addressed files, a nonzero *arg[0]* asserts the Remote Enable line and addresses the device. If *arg[0]* is zero, the device is removed from the remote state by sending it a Go-to-Local command (GTL).

`HPIB_SRQ` Request service. This request causes the interface to assert the Service Request line (SRQ) until it is serially polled. At that time it responds with the status byte given in `arg[1]`. This request applies to raw bus files only.

This request is normally used only when the interface is not the active controller. Nonetheless, the active controller can assert SRQ, and the `HPIB_BUS_STATUS` request will reflect the assertion; however, the SRQ line does not change state until the interface passes control.

`HPIB_PASS_CONTROL`
Pass active control of the bus. If the interface is currently active controller, this request relinquishes control of the bus, passing it instead to the device at the bus address in `arg[0]`. Passing control should be done with care, since it is not possible to detect whether the named device can indeed assume bus control. This request applies only to raw bus files.

`HPIB_GET_CONTROL`
Become active controller. This request causes the interface to assert Interface Clear (IFC) and Remote Enable (REN) as a means of regaining control of the HP-IB. It applies only to raw bus files.

Transparent Bus Request

This request allows a user to send direct commands over the HP-IB; it should be used with care, since improper use might place the bus in an unusable state.

The transparent bus request takes the following form:

```
struct hpib_command {
   int  length;
   char buffer[MAX_HPIB_COMMANDS];
} hpib_cmd;

ioctl(fildes, HPIB_COMMAND, &hpib_cmd);
```

This call transmits *length* bytes of data in *buffer* over the HP-IB with Attention (ATN) asserted. On completion of the request, ATN remains asserted.

For commands sent through an auto-addressed file, *buffer* is surrounded with the appropriate device addressing. What appears on the bus is:

UNL, TALK CIC, LISTEN device, *buffer*

This differs from the approach toward a raw bus file. For such files, the *buffer* is merely placed on the bus with ATN asserted, with no addressing or unaddressing.

Status Requests

These requests are used to obtain information about the general state of a device or the HP-IB. Their calling sequence is similar to that of control requests:

```
struct io_ctl_status hpib_status;
ioctl(fildes, IO_STATUS, &hpib_status);
```

As with the data structure to control requests, the **type** field specifies the type of information requested, while the **arg** array holds clarification data. The defined status requests for HP-IB and their use are described as follows:

HPIB_ADDRESS Return the bus address associated with the file in **arg[0]**.

HPIB_TIMEOUT Return the interface's timeout in microseconds in **arg[0]**.

HPIB_WIDTH Return the interface's path width in bits in **arg[0]**.

HPIB_SPEED Return the interface's data transfer rate in K-bytes per second in *arg[0]*.

HPIB_READ_PATTERN
Return the interface's read termination pattern in **arg[0]**; if pattern matching is not enabled, **arg[0]** holds a **-1**.

HPIB_SIGNAL_MASK
Return the reason for the last signal. This request returns a mask in **arg[0]**, with bits set indicating the reason(s) for the last **SIGEMT** sent to the user process. Bit definitions are identical to those of the corresponding **IO_CONTROL** request.

HPIB_LOCK Return lock status. If the device is locked to a process, return that process ID in **arg[0]** and the interface lock count in **arg[1]**. If the device is not locked, **arg[0]** holds a **-1**.

HPIB_TERM_REASON
Return end conditions for the last read from this device or bus. This request returns a byte in **arg[0]**, with a mask of reason(s) for the completion of the last read from the device or raw bus. Applicable bits are:

TR_COUNT	Read requested number of bytes.
TR_MATCH	Detected specified match pattern.
TR_TIMEOUT	Timed out.
TR_END	Device asserted EOI.
TR_ERROR	Detected bus error.
TR_NOTERM	No read done since open.

HPIB_PPOLL Conduct a parallel poll. This request returns the bus response to parallel poll in the least significant byte of **arg[0]**, with DIO1 corresponding to the least significant bit. The driver delays at least 100 microseconds before reading the poll response, thus allowing the use of **HPIB_PPOLL** on systems with extended buses. This request applies to both auto-addressed and raw bus files.

HPIB_SPOLL Conduct a serial poll. For raw bus files, this request conducts a serial poll of the device address in **arg[1]**; the status byte returned by the device is available in **arg[0]**. Auto-addressed files ignore any address in **arg[1]**, polling instead the device's predefined address.

HPIB_BUS_STATUS
Return the status of the HP-IB. This request, applicable to both types of files, returns information related to the current bus state. On return, **arg[0]** holds a value with bits set indicating:

ST_NDAC	NDAC is being asserted.
ST_SRQ	SRQ is being asserted.
ST_REN	Interface is in the remote state.
ST_ACTIVE_CTLR	Interface is active controller.

ST_SYSTEM_CTLR Interface is system controller.

ST_TALK Interface is addressed to talk.

ST_LISTEN Interface is addressed to listen.

ST_TALK_ALWAYS Interface is configured to talk always.

ST_LISTEN_ALWAYS
Interface is configured to listen always.

HPIB_WAIT_ON_PPOLL
Wait (sleep) until a given device responds to parallel poll. This request blocks the user until either the user's device responds to parallel poll (for auto-addressed files) or until any enabled devices respond (for raw bus files).

For a raw bus file, `arg[1]` and `arg[2]` contain eight-bit masks as defined in the HPIB_SIGNAL_MASK request. The return value of the request in `arg[0]` shows which devices responded to parallel poll.

For an auto-addressed file, `arg[1]` specifies the sense of the particular device's assertion. Successful completion of the request implies that the device responded.

HPIB_WAIT_ON_STATUS
Wait (sleep) until any of a set of given states is entered. The event(s) to await are specified by computing the bitwise inclusive OR of the values from the list below, and placing the mask in `arg[0]`. Applicable bits are:

ST_SRQ Wait until SRQ is asserted.

ST_ACTIVE_CTLR Wait until user is active controller.

ST_TALK Wait until user is addressed to talk.

ST_LISTEN Wait until user is addressed to listen.

Note that more than one bit can be set, thereby waiting for any of the events to occur. The return value in `arg[0]` is modified to show the actual event(s) that ended the wait. This is applicable to raw bus files only.

HPIB_INTERFACE_TYPE
Return the interface type. This returns one of two values in `arg[0]`:

HPIB_INTERFACE The open file is a HP-IB raw bus file.

HPIB_DEVICE The open file is a HP-IB auto-addressed file.

Extended Status Request

If the user wants to obtain several status variables in one request, the following request can be used:

```
struct io_environment hpib_env;

ioctl(fildes, IO_ENVIRONMENT, &hpib_env);
```

where the `io_environment` structure includes the following fields:

```
int interface_type;
int timeout;
int status;
int term_reason;
int read_pattern;
int signal_mask;
int width;
int speed;
int locking_pid;
```

Summary of Privilege Requirements

The following table summarizes which `ioctl()` requests can be performed under what circumstances (see *ioctl*(2)). The first three columns indicate whether the interface must be in the controlling state given to perform the request. An entry of Y means that the interface *must* be in that state, N means that the

interface must *not* be in that state, and – means that the state is irrelevant. The next two columns indicate whether the request works for auto-addressed or raw bus files. The final column indicates whether the request is subject to blocking while the interface is locked (see HPIB_LOCK).

If an entry is marked with an asterisk (*), check the particular request for more information.

Request	Non-Ctlr	Active Ctlr	System Ctlr	Auto Addr	Raw Bus	Lock Enforced
IO_CONTROL						
HPIB_TIMEOUT	–	–	–	Y	Y	N
HPIB_WIDTH	–	–	–	Y	Y	Y
HPIB_SPEED	–	–	–	N	Y	Y
HPIB_EOI	–	–	–	Y	Y	N
HPIB_SYSTEM_CTLR	–	–	–	N	Y	Y
HPIB_READ_PATTERN	–	–	–	Y	Y	N
HPIB_SIGNAL_MASK	*	*	–	Y	Y	Y
HPIB_LOCK	–	–	–	Y	Y	Y
HPIB_ADDRESS	–	–	–	N	Y	Y
HPIB_RESET						
DEVICE_CLR	N	Y	–	Y	N	Y
BUS_CLR	–	–	Y	Y	Y	Y
HW_CLR	–	–	–	N	Y	Y
HPIB_PPOLL_RESP	N	Y	–	Y	Y	Y
HPIB_PPOLL_IST	–	–	–	N	Y	Y
HPIB_REN	–	–	Y	Y	Y	Y
HPIB_SRQ	–	*	–	N	Y	Y
HPIB_PASS_CONTROL	N	Y	–	N	Y	Y
HPIB_GET_CONTROL	–	–	Y	N	Y	Y
HPIB_COMMAND	N	Y	–	Y	Y	Y
IO_STATUS						
HPIB_ADDRESS	–	–	–	Y	Y	Y
HPIB_TIMEOUT	–	–	–	Y	Y	N
HPIB_WIDTH	–	–	–	Y	Y	N
HPIB_SPEED	–	–	–	Y	Y	N
HPIB_READ_PATTERN	–	–	–	Y	Y	N
HPIB_SIGNAL_MASK	–	–	–	Y	Y	N
HPIB_LOCK	–	–	–	Y	Y	N
HPIB_TERM_REASON	–	–	–	Y	Y	N
HPIB_PPOLL	N	Y	–	Y	Y	Y
HPIB_SPOLL	N	Y	–	Y	Y	Y
HPIB_BUS_STATUS	–	–	–	Y	Y	Y
HPIB_WAIT_ON_PPOLL	N	Y	–	Y	Y	Y
HPIB_WAIT_ON_STATUS	–	–	–	N	Y	Y
HPIB_INTERFACE_TYPE	–	–	–	Y	Y	N
IO_ENVIRONMENT	–	–	–	Y	Y	Y

Default Configuration

The default configuration of any HP-IB file is:

Timeout	Infinite
Path Width	8 bits
Transfer Speed	0
EOI	Assertion Enabled
Pattern Match	Disabled
Enabled Signals	None
Locking	Unlocked

Termination Reason TR_NOTERM

ERRORS

A −1 return value for a driver request indicates that an error occurred; `errno` is set to specify the reason. In addition to errors defined in *open*(2), *close*(2), *read*(2), *write*(2), and *ioctl*(2), a driver request can fail if any of the following conditions are encountered:

[EACCES] The interface is not in the active-controller or system-controller state.

[EACCES] The interface is currently locked by another process via `HPIB_LOCK`.

[EACCES] A request to access the file would block and the `O_NDELAY` file status flag is set for the file descriptor.

[EINVAL] The request is not applicable to this type of file. Alternatively, *type* or *arg* has an invalid value.

[EINTR] An interface power failure occurred during the processing of this request; the device might have lost state.

[EINTR] A signal was received either while waiting for the interface to become unlocked, or while waiting for a `HPIB_WAIT_ON_PPOLL` or `HPIB_WAIT_ON_STATUS` request.

[EIO] An unclassified error occurred.

[EMFILE] The number of simultaneous `open()` requests on this interface exceeds the maximum allowed.

[ENXIO] No bus interface is associated with the device file.

[EPERM] An attempt was made to unlock an interface that was not locked.

[ERANGE] The interface lock count was exceeded.

[ETIMEDOUT] The transaction did not complete within the timeout specified.

In addition, the following messages can appear on the system console as a result of errors:

`instr0 unit` *%d*`: device adapter failure.`
The bus hardware is no longer functioning.

`instr0 unit` *%d*`: unexpected message (message type =` *%d*`, from port` *%d*`).`
The driver received an unclassifiable message.

WARNINGS

It is possible to circumvent the bus protection mechanisms afforded by the auto-addressed and raw bus dichotomy. Specifically, a user of an auto-addressed file can send commands to any or all devices on the bus with the `HPIB_COMMAND` request, if the proper device addressing is done within the data buffer.

The `HPIB_LOCK` request should be used with care. Since it provides an exclusive lock, invoking the `HPIB_LOCK` blocks access to any system disk or swap device on the associated bus.

Processes that use `HPIB_LOCK` should clear all locks before exiting. The driver attempts to clear them if the process terminates unexpectedly; however, a lock might be left outstanding if the locker dies after creating new file descriptors (via `fork()` or `dup()` that refer to the same device file (see *fork*(2) or *dup*(2)). Ensuring that all open file descriptors on a given interface are closed remedies the situation.

By default, some HP-IB peripherals respond to parallel poll on DIO*n*, where *n* has the value of 8 minus the device's bus address. That is, a device at address 6 can respond on DIO2. Therefore, the results of an `HPIB_PPOLL` request can be misleading if some devices are not remotely configured.

It is impossible to transfer data using a secondary address in a single driver request.

DEPENDENCIES

HP 27110B

The `HPIB_SPEED` and `HPIB_SYSTEM_CTLR` requests are not supported; they are configured by switches on the device adapter.

The `HPIB_SRQ` request can affect only the RQS bit of the serial poll response byte; all other bits are masked to zero by the hardware.

AUTHOR

`hpib` was developed by HP.

FILES

`/dev/hpib/*`

SEE ALSO

fcntl(5), ioctl(2), signal(2), sigvector(2), specific device documentation in section (7).

(Requires Optional LAN/X.25 Software)

NAME

inet - Internet protocol family

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
```

DESCRIPTION

The internet protocol family is a collection of protocols layered on top of the *Internet Protocol* (IP) network layer, which utilizes the internet address format. The internet family supports the SOCK_STREAM and SOCK_DGRAM socket types.

Addressing

Internet addresses are four byte entities. The include file `<netinet/in.h>` defines this address as a discriminated union, called `struct in_addr`.

Sockets bound to the internet protocol family utilize an addressing structure called `struct sockaddr_in`. Pointers to this structure can be used in system calls wherever they ask for a pointer to a `struct sockaddr`.

There are three fields of interest within this structure. The first is `sin_family`, which must be set to AF_INET. The next is `sin_port`, which specifies the port number to be used on the desired host. The third is `sin_addr`, which is of type `struct in_addr`, and specifies the address of the desired host.

Protocols

The internet protocol family is comprised of the IP network protocol, Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). TCP is used to support the **SOCK_STREAM** socket type while UDP is used to support the **SOCK_DGRAM** socket type. The ICMP message protocol and IP network protocol are not directly accessible.

The local port address is selected from independent domains for TCP and UDP sockets. This means that creating a TCP socket and binding it to local port number 10000, for example, does not interfere with creating a UDP socket and also binding it to local port number 10000 at the same time.

Port numbers in the range 1-1023 inclusive are reserved for use by the super-user only. Attempts to bind to port numbers in this range by non-super-users fail and result in an error returned.

AUTHOR

`inet` was developed by the University of California, Berkeley.

SEE ALSO

tcp(7P), udp(7P).

NAME

iomap - physical address mapping

SYNOPSIS

```
#include <sys/iomap.h>
```

DESCRIPTION

The `iomap` mechanism allows the mapping (thus direct access) of physical addresses into the user process address space. For most Series 300/400 computers, the physical address space begins at `0x000000` and extends to `0xffffff`.

The special (device) files for `iomap` devices are character special files with major number 10.

The minor number for `iomap` devices is of the form:

0xAAAANN

where AAAA is a two-byte address, and NN is a one-byte field.

The address portion of the minor number is formed by dividing the physical address by 65 536. NN × 65 536 is the size of the region to be mapped. For example, the minor number for a device at `0x720000` that occupies 128 Kbytes is `0x007202`.

Access to `iomap` devices is controlled by file permissions set on the character special file.

Multiple processes can concurrently have a single `iomap` device opened and mapped. It is the responsibility of the processes to synchronize their access.

No `read()` or `write()` system calls are supported by the `iomap` driver.

The `ioctl()` function is used to control the `iomap` device. The following `ioctl()` requests are defined in <`iomap.h`>:

`IOMAPMAP` Map the `iomap` device into user address space at the location specified by the pointer to which the `(void **)` third argument to `ioctl()` points. If the argument points to a variable containing a null pointer, the system selects an appropriate address. `ioctl()` then returns the user address where the device was mapped, storing it at the address pointed to by the third argument (see EXAMPLES below). Multiple processes can concurrently have the same `iomap` device mapped.

`IOMAPUNMAP` Unmap the `iomap` device from the user address space.

`close()` shuts down the file descriptor associated with the `iomap` device. If the close is for the last system wide open on the device, the `iomap` device is also unmapped from the user address space; otherwise it is left mapped into the user address space (see `IOMAPUNMAP` above).

One shared memory descriptor (see *shmget*(2)) is used for each `iomap` device. Shared memory descriptors obtained in this way are usable only through the `iomap` interface. Consequently, attempts to access them through shared memory routines (see *shmat*(2), *shmctl*(2), *shmdt*(2), etc.) result in EACCES errors.

WARNING

Be extremely careful when creating and using `iomap` devices. Inappropriate accesses to I/O devices or RAM can result in a system crash.

ERRORS

[EINVAL] The address field was out of range, or the `ioctl` request was invalid.

[ENOMEM] Not enough memory could be allocated for the mapping.

[ENODEV] Read and write calls are unsupported.

[ENXIO] No such device at the address specified by the minor number.

[ENOSPC] Required resources for mapping could not be allocated.

[ENOTTY] Inappropriate `ioctl` request for this device type; *fildes* is not a file descriptor for an `iomap` device file.

EXAMPLES

Consider the following code fragment:

```
#include <sys/iomap.h>
    ...
int fildes;
void *addr;

    ...
    addr = REQUESTED_ADDRESS;
    (void) ioctl(fildes, IOMAPMAP, &addr);
    (void) printf("actual address = 0x%x\n", addr);
```

where `fildes` is an open file descriptor for the device special file and `REQUESTED_ADDRESS` is the address originally requested by the program.

If `addr` is a null pointer, the system selects a suitable address then returns the selected address in *addr*.

If the value in *addr* is not a null pointer, it is used as a specified address for allocating memory. If the specified address cannot be used, an error is returned (see ERRORS).

SEE ALSO

mknod(1M).

NAME

lan - network I/O card access information

DESCRIPTION

This manual entry explains how to access the LAN device driver at Layer 2 (Data Link Layer) of the OSI architecture. The LAN device driver controls the Ethernet/IEEE 802.3 LAN interface card at Layer 1 (Physical Layer).

Link Level Access is intended for use by knowledgeable network users only. Refer to the *LLA Programmer's Guide* for complete programming details.

System Calls

The following system calls are used to access the driver that controls the IEEE 802.3/Ethernet interface card:

`open()` Opens the device file associated with the driver. The only applicable option flags are the delay flag, **O_NDELAY**, the read only flag, **O_RDONLY**, and the read/write flag, **O_RDWR**. If **O_NDELAY** is set and no data is available, a `read()` call returns immediately. If you want to use only the **NETSTAT** commands, specify the **O_RDONLY** flag. Otherwise, the **O_RDWR** flag must be specified. These flags are defined in the header <**sys/fcntl.h**>.

`close()` Closes a network device file.

`read()` Reads data from the network. The maximum number of bytes that can be transferred per `read()` call is 1500 for Ethernet and 1497 for IEEE 802.3.

Read operations may block (if **O_NDELAY** was not specified in open) if system resources are not immediately available to perform the operation. Blocked read operations terminate upon delivery of signals to the calling process.

Read and write operations can only address a single packet of data appropriate for the protocol being used.

`write()` Writes data out to the network. The maximum number of bytes that can be transferred per `write()` call is 1500 for Ethernet and 1497 for IEEE 802.3.

Write operations never block, returning instead an error indicating the type of resource limitation that prevents performing the operation (see for specific details).

Read and write operations can only address a single packet of data appropriate for the protocol being used.

`select()` Used before `read()` and `write()` calls to help an application synchronize its I/O operations. `select()` is supported for read and write operations, but it is not supported for exceptional conditions.

`ioctl()` Used to construct, inspect, and control the network environment in which the application operates. All applications must use the `ioctl()` call to configure source and destination addresses before data can be sent or received using `read()` and `write()` calls. The `ioctl()` syntax used for these operations is as follows:

```
#include <netio.h>

ioctl (fildes,request,arg);
int fildes,request;
struct fis *arg;
```

Parameters in the calling sequence are:

fildes The file descriptor of the successfully opened Ethernet/IEEE 802.3 device.

request Specifies which type of command to perform. This parameter must be either **NETSTAT** or **NETCTRL**.

arg Address of the `fis` data structure. The `fis` data structure contains information necessary to perform a specific **NETCTRL** or **NETSTAT** command. The *arg* parameter must be set to the address of a `fis` structure before an `ioctl()` call is made. The type of information stored in *arg* is:

```
struct fis { int reqtype;
     int vtype;
     union { float f;
          int i;
          unsigned char s[100];
          } value;
     };
```

where:

`reqtype` contains the name of the **NETCTRL** or **NETSTAT** command to be executed.

`vtype` identifies the type of the value in the `value` union. If `vtype = INTEGERTYPE`, the value is in `value.i`. If `vtype = FLOATTYPE`, the value is in `value.f`. If `vtype` = a non-negative integer, the value is a character string in `value.s`. This integer also specifies the length of the string. The `vtype` parameter must only be specified for **NETCTRL** commands.

NETCTRL and NETSTAT Commands

NETCTRL commands are used to set up device-specific parameters prior to read and write operations, and to reset the network I/O card and its statistical registers. There are two types of **NETCTRL** commands: those that affect the network I/O cards and those that affect a particular connection to the network I/O card.

NETSTAT commands are used to obtain device-dependent status and statistical information.

Station Management NETCTRL Commands

The following **NETCTRL** `arg.reqtype` arguments control the read/write activities of the network device on a per-file-descriptor basis:

`LOG_TYPE_FIELD`

This command is restricted to the Ethernet protocol, and corresponds to the *type* field of the Ethernet header. When co-existing on a multi-vendor network, the user should contact Xerox Corporation for authorization to use a given *type* field. See **LOG_DSAP** and **LOG_SSAP** below for the corresponding IEEE 802.3 commands.

Before performing `read()` and `write()` operations, a *type* field must be logged with the driver. This *type* field is the effective *user address* for the network connection being established, and must be unique among all open network connections on the same interface card or an error results. Only one *type* field can be declared per (open) file descriptor; once logged, it cannot be changed.

The format of the *type* field required by `ioctl()` is an integer in the range 2048..65535.

LOG_TYPE_FIELD is the first call made in establishing a network connection. The call fails if memory needed to log the *type* field is unavailable.

The device header prefixed to the user data by the driver on each `write()` call contains this *type* field. `read()` calls return the data portion of only those packets whose *type* field in the device header matches the logged *type* field.

The following type fields are reserved addresses: 2048, 2053, 2054, 32773. Any attempt to log these types fails, returning EBUSY to the calling process.

Other specifically reserved addresses include 4096 through 4111. These types are reserved for use by Berkeley Trailer Protocols.

The following parameters must be set prior to calling `ioctl()`:

```
arg.vtype     = INTEGERTYPE;
arg.value.i = type field ;
```

LOG_SSAP

This command applies only to the IEEE 802.3 protocol.

Before performing `read()` and `write()` operations, a source service access point (SSAP) must be logged with the driver. This value is the effective *user address* for the network connection being established, and must be unique among all open network connections on the same interface card or an error results. Only one SSAP can be declared per (open) file descriptor. Once logged, it cannot be changed.

The format of the SSAP required by `ioctl()` is an even integer in the range 2..254.

LOG_SSAP is the first call made in establishing a network connection; the call fails if memory needed to log the SSAP is unavailable.

LOG_SSAP sets the destination service access point (DSAP) value to the SSAP value. Under normal circumstances, DSAP and SSAP should remain equal.

The following SSAP values are reserved addresses: 6, 252, 248. An attempt to log these SSAPS fail with EBUSY returned to the calling process.

The device header inserted in front of the user data by the driver on each `write()` call contains this SSAP/DSAP. `read()` calls return the data portion of only those received packets whose DSAP value in the device header matches the logged SSAP value.

The following parameters must be set prior to calling `ioctl()`:

```
arg.vtype = INTEGERTYPE;
arg.value.i = SSAP;
```

LOG_DSAP

This command applies only to the IEEE 802.3 protocol.

LOG_DSAP can be called to change the default DSAP value set up by LOG_SSAP. The DSAP value can be changed any time after the LOG_SSAP call and can be changed any number of times.

The format of the DSAP required by `ioctl()` is an integer in the range 0..254.

The device header inserted in front of the user data by the driver on each `write()` call contains this DSAP.

The following parameters must be set prior to calling `ioctl()` :

```
arg.vtype = INTEGERTYPE;
arg.value.i = DSAP ;
```

LOG_DEST_ADDR

Before performing `write()` operations, a destination address must be logged with the driver. This address is the network station address of the remote Ethernet/IEEE 802.3 device that is to receive the data. LOG_DEST_ADDR commands can be issued any time but must be specified prior to attempting a write. LOG_DEST_ADDR commands can be issued any number of times during a LLA connection.

A header inserted in front of the user data on each `write()` call contains this *destination address*.

The following parameters must be set prior to calling `ioctl()`:

```
arg.vtype = 6;
arg.value.s = destination address ;
```

LOG_READ_TIMEOUT

The default (0) timeout value blocks a user process executing a `read()` until data is available (see LOG_TYPE_FIELD). A positive timeout value causes the `read()` to fail after the specified time has elapsed if no data is available. A negative timeout value fails with EINVAL

returned. If the O_NDELAY flag is set, the timeout mechanism is overridden and the error, EWOULDBLOCK, is returned to the calling process.

Due to race conditions caused by asynchronous interrupts, the accuracy of the timer is guaranteed only to the extent that it will not timeout sooner than the assigned value.

The following parameters must be set prior to calling `ioctl()`:

```
arg.vtype = INTEGERTYPE;
arg.value.i = read timeout value in milliseconds ;
```

LOG_READ_CACHE

By default, only one packet received for an active type field or DSAP is cached, pending a `read()` to the associated file descriptor. Subsequent packets received for that file descriptor are discarded. This one-packet cache is suitable for request/reply protocols, but may not be suitable for applications that communicate with more than one host or where windowing protocols are used. LOG_READ_CACHE can be used to increase the the receive cache buffer to a total of 16 packets for a normal user and a total of 64 packets for a user with appropriate privileges. Any request for additional packet caching that would result in a cache size greater than the above limits is truncated to the appropriate limit. The following parameters must be set prior to calling `ioctl()`:

```
arg.vtype = INTEGERTYPE;
arg.value.i = additional packets to cache ;
```

LOG_CONTROL

This command applies only to the IEEE 802.3 protocol and conforms to its specification. Refer to the IEEE 802.3 specification for detailed information about the UI, XID, and TEST control fields mentioned below.

This command requires appropriate privileges.

The Unnumbered Information (UI) control field of the IEEE 802.3 header is the default used for normal communication. This default can be overridden with XID_CONTROL or TEST_CONTROL. When using either the XID or TEST control field, any data written to the network device is ignored; an XID or TEST request packet is transmitted instead, and any response from other network nodes is returned via the `read()` call.

Prior to calling `ioctl()` , the `arg.vtype` parameter must be set to INTEGERTYPE; the `arg.value.i` parameter must be set to UI_CONTROL, XID_CONTROL, or TEST_CONTROL.

LLA_SIGNAL_MASK

This request allows the user to specify which defined LAN events can generate a SIGIO signal to the process which configured it. Currently, the LAN events which can generate SIGIO are receipt of packet on LLA connection and inbound queue overflow for LLA connection. Mask values can be derived by ORing one or more of the following LAN events into the `value.i` operand:

LLA_PKT_RECV

is used when the process is to be signaled every time a packet is successfully queued for subsequent reading. If the queue is full and the packet must be discarded, no signal is generated. Also, if the process is blocked on a read to the LLA connection, no signal will be generated because this would interfere with the current read.

LLA_Q_OVERFLOW

is used when the process is to be signaled every time a packet must be discarded as a result of inbound queue overflow.

To eliminate signal generation completely, LLA_NO_SIGNAL should be assigned (not ORed) to `value.i`.

Be careful when combining mask values because receipt of a signal does not convey which event occurred. Note also that the driver signals only that process which *last* configured the LLA_SIGNAL_MASK. Processes that share file descriptors can potentially interfere with the intended use of LLA SIGIO.

Prior to calling `ioctl()`, the `arg.vtype` parameter must be set to `INTEGER-TYPE`; the `arg.value.i` parameter must be set to either `LLA_NO_SIGNAL` or a combination of `LLA_PKT_RECV`, and `LLA_Q_OVERFLOW`.

Device Control NETCTRL Commands
The following are global `NETCTRL arg.reqtype` arguments that affect the Ethernet/IEEE 802.3 device itself, as opposed to a particular open file descriptor associated with the device, and are accessible only to users with appropriate privileges.

`RESET_INTERFACE`
Resets the Ethernet/IEEE 802.3 device, forcing a complete hardware self-test. All statistical counters are reset to zero. No parameters are necessary. The current multicast state is retained.

A reset can drop packets or impair any currently active network connection to the local computer.

`ADD_MULTICAST`
Adds a multicast address to the device's list of accepted multicast addresses. A received packet is discarded if its multicast address is not in the current list of accepted multicast addresses. A multicast address is a 48-bit value with the least significant bit of the first octet set to indicate a group address. A multicast address list can contain up to 16 addresses.

The broadcast address is not handled as a typical multicast address and is documented below under ENABLE_BROADCAST/DISABLE_BROADCAST.

The following parameters must be set prior to calling `ioctl()`:

```
arg.vtype = 6;
arg.value.s = multicast address ;
```

`DELETE_MULTICAST`
Removes a specified multicast address from the current list of accepted multicast addresses.

The following parameters must be set prior to calling `ioctl()`:

```
arg.vtype = 6;
arg.value.s = multicast address ;
```

Deletion of any HP-special multicast address may prove catastrophic to an active HP network. The HP multicast addresses are: 0x090009000001, 0x090009000002.

`ENABLE_BROADCAST`
Allows broadcast packets to be received by the network device. No parameters are needed.

`DISABLE_BROADCAST`
Disallows broadcast packets from being received by the network device. No parameters are needed.

Users with appropriate privileges are cautioned that use of this command may be catastrophic to an active HP network.

Reset and Read Statistics Commands
The following commands are provided to collect and reset interface statistics and can be used as `NETCTRL` or `NETCTRL ioctl()` commands. All of the `NETCTRL` commands require appropriate privileges. When *request* equals `NETCTRL` and `arg.reqtype` equals `RESET_STATISTICS`, *all* interface statistics counters are reset to zero. When *request* equals `NETSTAT`, the current value of the statistic specified in `arg.reqtype` is returned. The following section assumes that the specified *request* is `NETSTAT`.

`RX_FRAME_COUNT` Returns the number of packets received without error.

`TX_FRAME_COUNT` Returns the number of packets transmitted without error.

UNTRANS_FRAMES Returns the number of packets that, due to some error, could not be transmitted.

UNDEL_RX_FRAMES
Returns the number of packets which were received, but due to some error, could not be delivered to an appropriate network connection.

RX_BAD_CRC_FRAMES
Returns the number of packets received with a bad CRC.

COLLISIONS Returns the number of transmit attempts that were retransmitted due to collisons.

RESET_STATISTICS
Not applicable.

DEFERRED Returns the number of packets that had to defer before transmission.

ONE_COLLISION Returns the number of transmissions completed with one collision.

MORE_COLLISIONS
Returns the number of transmissions completed with more than one collision.

EXCESS_RETRIES Returns the number of packets that were not transmitted due to an excessive number of retries (16 or more).

LATE_COLLISIONS
Returns the number of transmit packets for which the card detected a late collision.

CARRIER_LOST Returns the number of transmit packets that failed due to the loss of the carrier. This is a hardware-dependent statistic that indicates problems with the Medium Attachment Unit (MAU) cabling.

NO_HEARTBEAT Returns the number of transmit packets for which no heartbeat was detected. This is a hardware-dependent statistic that indicates problems with the Medium Attachment Unit (MAU) cabling.

ALIGNMENT_ERRORS
Returns the number of packets received with an alignment error and a bad CRC. These packets are also counted in the **RX_BAD_CRC_FRAMES** count.

MISSED_FRAMES Returns the number of times that the card missed packets due to lack of resources.

BAD_CONTROL_FIELD
Returns the number of IEEE 802.3 packets received with an invalid control field.

UNKNOWN_PROTOCOL
Returns the number of packets dropped because the type field or DSAP referenced an unknown protocol.

RX_XID Returns the number of IEEE 802.3 XID packets that were received.

Refer to the discussion of the **LOG_CONTROL** command earlier on this reference page for more information about XID packets.

RX_TEST Returns the number of IEEE 802.3 TEST packets that were received.

RX_SPECIAL_DROPPED
Returns the number of IEEE 802.3 XID or TEST packets that were received but not responded to due to lack of space on the outbound queue.

NO_TX_SPACE Returns the number of times that the card exhuasted its transmit buffer space. This statistic is kept by Series 800 systems only.

LITTLE_RX_SPACE
Returns the number of times the card had one or no buffers to accept incoming packets. This statistic is kept by Series 800 systems only.

ILLEGAL_FRAME_SIZE
Returns the number of times the card received and discarded packets that were illegal in size (greater than 1514 bytes). This statistic is kept by Series 800 systems only.

TDR
Returns the time (in bit times) from when a frame started to transmit until a collision occurred. This statistic can be useful for grossly determining where on the cable a problem is located. This statistic is not updated after an external loopback frame is transmitted. This statistic is kept by Series 800 systems only.

The following commands are provided to collect interface statistics, but can be only used as **NETSTAT** commands.

FRAME_HEADER
Return the Ethernet/IEEE 802.3 device header associated with the previous **read()** call. The result of a FRAME_HEADER call is undefined if there has been no previous **read()**.

For Ethernet, **arg.vtype** is 14, and **arg.value.s** holds the following information:

s[0]..s[5] holds the destination address, as determined by the sender; this address could be the local Ethernet device's network station address, or a multicast/broadcast address;

s[6]..s[11] holds the network station address of the sender's Ethernet device (i.e. source address);

s[12]..s[13] holds the type field (user's address) as a 16-bit unsigned integer.

For IEEE 802.3, **arg.vtype** is 17, and **arg.value.s** holds the following information:

s[0]..s[5] holds the destination address, as determined by the sender. This address could be the local IEEE 802.3 device's network station address or a multicast/broadcast address.

s[6]..s[11] holds the network station address of the sender's IEEE 802.3 device (i.e. source address).

s[12]..s[13] holds the received packet's length, including data, the SSAP/DSAP and the control field.

s[14] holds the destination sap value;

s[15] holds the source sap value;

s[16] holds the control field value.

LOCAL_ADDRESS
Returns the address of the local Ethernet/IEEE 802.3 device in **arg.value.s**. **arg.vtype** is 6 (length of **arg.value.s**). Any byte of the address can be NULL.

DEVICE_STATUS
Returns the current status of the Ethernet/IEEE 802.3 device. Possible return values are **INACTIVE**, **INITIALIZING**, **ACTIVE** and **FAILED**.

MULTICAST_ADDRESSES
Returns the current number of accepted multicast addresses in **arg.value.i**.

MULTICAST_ADDR_LIST
Returns the current list of accepted multicast addresses in **arg.value.s**. The value specified in **arg.vtype.s** represents the number of bytes used for the contiguous address list in **arg.value.s**. Each address is six bytes long. The maximum number of bytes that can be returned is 96.

DIAGNOSTICS

If an error occurs, the error value is given in `errno`.

[EIO] `read()/write()` failure (includes timeout conditions).

[EPERM] A user attempted to call a command that requires privileges the user does not have.

[ENOMEM] Series 300 only. One of the following:

- Memory requested by `LOG_READ_CACHE` is unavailable.
- Memory requested by `LOG_TYPE_FIELD` is unavailable.
- Memory requested by `LOG_SSAP` is unavailable.
- Memory requested by a `write()` call is unavailable.

[EBADF]
An attempt was made to write (or `NETCTRL ioctl()`) to a device that was opened with `O_RDONLY` permission.

[EBUSY]
An attempt was made to log a user-level address that is already in use.

[ENXIO]
One of the following:

- An attempt was made to `open()` LAN device with incorrect select code (Series 300/400) or incorrect logical unit or protocol (Series 700/800). `lanscan` can be used to display the select code (Series 300/400) or the hardware path (Series 700/800) and the logical unit of each LAN interface card (see *lanscan*(1M)).
- The specified driver call could not complete because the interface card was found to be in a `DEAD` state. The card must be reset before any further interface activity can resume.

[EINTR]
During a blocked read, the calling process was delivered a software interrupt prior to receiving a packet on its inbound queue.

[EDESTADDRREQ]
`read()/write()` call preceded a `LOG_TYPE_FIELD` or `LOG_SSAP` call.

`write()` call preceded a `LOG_DEST_ADDR` call.

[EMSGSIZE]
An attempt was made to `write()` more than the maximum bytes allowed by the selected protocol.

[EINVAL]
One of the following:

- An attempt was made to `write()` or `read()` a negative number of bytes.
- An attempt was made to `open()` with bad *oflag* value.
- `LOG_DSAP` call preceded a `LOG_SSAP` call.
- `LOG_TYPE_FIELD` call was sent to an IEEE 802.3 device.
- `LOG_SSAP`, `LOG_DSAP`, `LOG_CONTROL`, or RX_XID , `RX_TEST`, `RX_SPECIAL_DROPPED`, `BAD_CONTROL_FIELD` calls were sent to an Ethernet device.
- An attempt was made to log a user address and the SSAP or TYPE was out of range.
- An attempt was made to change a type_field or SSAP (user-level address).
- Improper address format in an `ioctl()` call involving an address.
- An `ADD_MULTICAST` call was attempted but the supplied address was already in the list.

- An **ADD_MULTICAST** call was attempted but 16 multicast addresses were already logged (list full).
- A **DELETE_MULTICAST** call was attempted but the supplied address was not in the list.
- A **DELETE_MULTICAST** call was attempted but no multicast addresses have been logged (list empty).
- An **ADD_MULTICAST** or **DELETE_MULTICAST** call was attempted but the multicast bit was not set in address operand.
- Timeout value passed to **LOG_READ_TIMEOUT** was negative.
- Unknown **arg.reqtype**.
- Incorrect **arg.vtype**.
- *fildes* does not specify an active network I/O device.
- An attempt was made to set **LLA_SIGNAL_MASK** with undefined events set in the mask operand.

[ENOSPC]
An attempt to **write()** a packet failed as a result of an outbound queue overflow on the interface card.

[ENOBUFS]
An **open()**, **read()**, **write()**, or **ioctl()** call could not get enough memory.

NOTES

The **fstat()** system call does not support LAN device files (see *fstat*(2)). When **fstat()** is called on a LAN device file, the error code [EINVAL] is returned. Use **stat()** instead (see *stat*(2)).

WARNINGS

Multiple processes sharing a file descriptor can interfere with each other, especially with respect to the commands **LOG_READ_TIMEOUT**, **LOG_READ_CACHE**, **LOG_DEST_ADDR**, and **LLA_SIGNAL_MASK**.

Also, the network driver does not guarantee data delivery. On a successful **write()**, the only guarantee is that data has been queued for transmission by the network I/O card. Likewise, there is no guarantee that once transmitted, data is received by the target node. The desired degree of reliability must be coded into the user program.

AUTHOR

lan was developed by HP

SEE ALSO

lanscan(1M), close(2), fcntl(2), ioctl(2), open(2), read(2), select(2), signal(2), write(2), net_aton(3N).

The Ethernet, A LAN: Data Link Layer and Physical Specification, Version 2.0, November 1982, Digital Equipment Corporation, Intel Corporation, Xerox Corporation

CSMA/CD Access Method and Physical Specification, October 1984, Institute of Electrical and Electronic Engineers

LLA Programmer's Guide.

DEPENDENCIES

Series 300

NO_TX_SPACE, **LITTLE_RX_SPACE**, **TDR**, **RX_XID**, **RX_TEST**, **RX_SPECIAL_DROPPED**, **MULTICAST_ADDR_LIST**, and **ILLEGAL_FRAME_SIZE** statistics are not kept by Series 300 systems.

NAME

lp - line printer

Remarks:

This manual entry applies only to a certain group of printers. For Series 800 systems, it applies to printers controlled by the device drivers **lpr0**, **lpr1**, and **lpr2**. For Series 300/400 systems, it applies to printers controlled by device drivers **printer** and **ciper**. It does *not* apply to any printers on Series 700 systems.

SYNOPSIS

#include <sys/lprio.h>

DESCRIPTION

This section describes capabilities provided by many line printers supported by various versions of the HP-UX operating system. A line printer is a character special device that may optionally have an interpretation applied to the data.

If the character special device file has been created with the raw option (see the HP-UX System Administrator manuals for information about creating device files with the raw option), data is sent to the printer in *raw mode* (as, for example, when handling a graphics printing operation). In raw mode, no interpretation is done on the data to be printed, and no page formatting is performed. Data bytes are simply sent to the printer and printed exactly as received.

If the device file does not contain the raw option, data can still be sent to the printer in raw mode. Raw mode is set and cleared by the LPRSET request.

If the line printer device file does not contain the raw option, data is interpreted according to rules discussed below. The driver understands the concept of a printer page in that it has a page length (in lines), line length (in characters), and offset from the left margin (in characters). The default line length, indent, lines per page, open and close page eject, and handling of backspace are set to defaults determined when the printer is opened and recognized by the system the first time. If the printer is not recognized, the default line length is 132 characters, indent is 4 characters, lines per page is 66, one page is ejected on close and none on open, and backspace is handled for a character printer.

The following rules describe the interpretation of the data stream:

- A form feed causes a page eject and resets the line counter to zero.
- Multiple consecutive form-feeds are treated as a single form-feed.
- The new-line character is mapped into a carriage-return/line-feed sequence, and if an offset is specified a number of blanks are inserted after the carriage-return/line-feed sequence.
- A new-line that extends over the end of a page is turned into a form-feed.
- Tab characters are expanded into the appropriate number of blanks (tab stops are assumed to occur every eight character positions as offset by the current indent value).
- Backspaces are interpreted to yield the appropriate overstrike either for a character printer or a line printer.
- Lines longer than the line length minus the indent (i.e., 128 characters, using the above defaults) are truncated.
- Carriage-return characters cause the line to be overstruck.
- When it is opened or closed, a suitable number of page ejects is generated.

Two *ioctl*(2) requests are available to control the lines per page, characters per line, indent, handling of backspaces, and number of pages to be ejected at open and close times. At either open or close time, if no page eject is requested the paper will not be moved. For opens, line and page counting will start assuming a top-of-form condition.

The *ioctl* requests have the following form:

#include <sys/lprio.h>

int ioctl(int fildes, int request, struct lprio *arg);

The possible values of *request* are:

LPRGET Get the current printer status information and store in the **lprio** structure to which *arg* points.

LPRSET Set the current printer status information from the structure to which *arg* points.

The **lprio** structure used in the LPRGET and LPRSET requests is defined in **<sys/lprio.h>**, and includes the following members:

```
short int   ind;        /* indent */
short int   col;        /* columns per page */
short int   line;       /* lines per page */
short int   bksp;       /* backspace handling flag */
short int   open_ej;    /* pages to eject on open */
short int   close_ej;   /* pages to eject on close */
short int   raw_mode;   /* raw mode flag */
```

These are remembered across opens, so the indent, page width, and page length can be set with an external program. If the **col** field is set to zero, the defaults are restored at the next open.

If the backspace handling flag is 0, a character printer is assumed and backspaces are passed through the driver unchanged. If the flag is a 1, a line printer is assumed, and sufficient print operations are generated to generate the appropriate overstruck characters.

If the raw mode flag is 0, data sent to the printer is formatted according to indent, columns per page, lines per page, backspace handling, and pages to eject on open and close.

If the raw mode flag is 1, data sent to the printer is not formatted.

If the raw mode flag is changed from 1 to 0 (raw mode is turned off) and the format settings (indent, columns per page, etc.) have not been modified, the data is formatted according to the prior format settings.

DEPENDENCIES

Series 300/400

The uppercase-only flag, the no-overprint flag, the raw-mode flag, and no-page-eject-on-open-or-close flag can be selected (enabled) by appropriate use of the minor number in the *mknod*(1M) command. See the HP-UX System Administrator manuals for details.

AUTHOR

lp was developed by HP and AT&T.

FILES

/dev/lp default or standard printer used by some HP-UX commands;

/dev/[r]lp* special files for printers

SEE ALSO

lp(1), slp(1), ioctl(2), cent(7), intro(7).

NAME

mem, kmem - main memory

DESCRIPTION

mem is a special file that is an image of the main memory of the computer. It may be used, for example, to examine and patch the system.

Byte addresses in *mem* are interpreted as physical memory addresses. References to non-existent locations cause errors to be returned.

File *kmem* is the same as *mem* except that kernel virtual memory rather than physical memory is accessed.

WARNINGS

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

FILES

/dev/mem
/dev/kmem

NAME

modem - asynchronous serial modem line control

SYNOPSIS

#include <sys/modem.h>

DESCRIPTION

This section describes the two modes of modem line control and the three types of terminal port access. It also discusses the effect of the bits of the *termio* structure that affect modem line control. The modem related *ioctl*(2) system calls are discussed at the end of the document.

Definitions

There are several terms that are used within the following discussion which will be defined here for reference. "Modem control lines" (CONTROL) are generally defined as those outgoing modem lines that are automatically controlled by the driver. "Modem status lines" (STATUS) are generally defined as those incoming modem lines that are automatically monitored by the driver. CONTROL and STATUS for a terminal file vary according to the modem line control mode of the file (see **Modem line control modes** below). An *open*(2) to a port will be considered to be BLOCKED if it is waiting for another file on the same port to be closed. An *open* to a port is considered to be PENDING if it is waiting for the STATUS to be raised. An *open* to a port is considered to be SUCCESSFUL if the *open* system call has returned to the calling process without error.

Open flag bits

Currently, the only *open* flag bits recognized by the driver is the O_NDELAY and O_NONBLOCK bits. When either of these bits is set, an *open* call to the driver will never become blocked. If possible, the *open* will be returned immediately as SUCCESSFUL, and the driver will continue the process of opening the tty file. If it is not possible, then the *open* will be returned immediately with the appropriate error code as described in the appropriate section.

Termio bits

When set, the CLOCAL bit in the *termios* or *termio* structure (see *termio*(7)) is used to remove the driver's automatic monitoring of the modem lines. However, the user's ability to control the modem lines is determined only by the mode in effect and does not depend on the state of CLOCAL. Normally, the driver will monitor and require the STATUS to be raised. An *open* system call will raise the CONTROL and wait for the STATUS before completing unless the CLOCAL bit is set. (If the O_NDELAY or O_NONBLOCK bit is set, the *open* will be returned immediately, but the driver will otherwise continue to monitor the modem lines as normal based on the state of the CLOCAL bit.) Normally, loss of the STATUS will cause the driver to break the modem connection and lower the CONTROL. However, if CLOCAL is set, any changes in the STATUS will be ignored. A connection is required before any data may be read or written, unless CLOCAL is set. Any timers that would normally be in effect (see **Modem line control modes** and **Modem timers** below) will be stopped while CLOCAL is set.

When the CLOCAL bit is changed from clear to set, the driver will assume the existence of an active device (such as a modem) on the port regardless of the STATUS. If any of the CONTROL are raised at that point in time, they will continue in that state. The STATUS will no longer be actively monitored. When the CLOCAL bit is changed from set to clear, the driver will resume actively monitoring the STATUS. If all of the CONTROL and STATUS are raised at that point in time, the driver will continue the modem connection. If any of the STATUS are not raised, the driver will act as though those signals were lost (as described in **Modem line control modes** below) and, if the device is a controlling terminal, a *hangup* signal will be sent to the controlling process. If any of the CONTROL are not raised, the driver will break the modem connection by lowering all the CONTROL.

The HUPCL bit in the *termios* or *termio* structure determines the action of the driver regarding the CONTROL when the last *close* system call is issued to a terminal file. If the HUPCL bit is set, the driver will lower the CONTROL at *close* time and the modem connection will be broken. If HUPCL is not set and a modem connection exists, it will continue to exist, even after the *close* is issued. The driver will not change the CONTROL.

Terminal port access types

There are three types of modem access: call-in connections, call-out connections, and direct (no modem control) connections. A given port may be accessed through all three types of connection by accessing different files. The modem access type of a terminal file is determined by the file's major and/or minor device numbers.

The call-in type of access is used when the connection is expected to be established by an incoming call. This is the type that would be used by *getty*(1M) to accept logins over a modem. When an *open* is issued to such a file, the driver may wait for an incoming call and will then raise the CONTROL based on the current mode (see below) of the port. When the port is closed, the driver may or may not lower the CONTROL depending on the HUPCL bit.

The call-out type of access is used when the connection is expected to be established by an outgoing call. This would be used by programs such as *uucp*(1). When an *open* is issued to such a file, the driver will immediately raise the CONTROL and wait for a connection based on the mode currently in effect. When the port is closed, the driver may or may not lower the CONTROL depending on the HUPCL bit.

The direct type of access is used when no driver modem control is desired. This could then be used for directly connected terminals that use a three-wire connection, or to talk to a modem before a connection has been established. The second case allows a program to give dialing instructions to the modem. Neither the CLOCAL nor the HUPCL bits have any effect on a port accessed through a direct file. (However, both bits may be inherited by other types of files; see **Terminal port access interlock** below.) An *open* to a direct file does not affect the CONTROL and does not depend on any particular state of the STATUS to succeed. When the file is closed, the driver will not affect the state of the CONTROL. If a modem connection has been established, it will continue to exist. Setting the speed of a direct file to B0 (see *termio*(7)) will be considered an impossible speed change and will be ignored. It will not affect the CONTROL.

Modem line control modes

There are two modes of modem line control: CCITT mode and simple mode. A given port may have only one of these two modes in effect at any given point in time. An attempt to *open* a port with a mode other than the one in effect (from a PENDING or SUCCESSFUL *open* on a different file) will cause the *open* to be returned with an ENXIO error. The modem access type of a terminal file is determined by the file's major and/or minor device numbers.

CCITT mode is used for connections to switched line modems. The CONTROL for CCITT mode are Data Terminal Ready and Request to Send The STATUS are Data Set Ready (DSR), Data Carrier Detect (DCD), and Clear to Send (CTS). Additionally, the Ring Indicator (RI) signal indicates the presence of an incoming call. When a connection is begun (an incoming call for a call-in file or an *open* issued to a call-out file), the CONTROL are raised and a connection timer (see **Modem timers** below) is started. If the STATUS become raised before the time period has elapsed, a connection is established and the *open* request is returned successfully. If the time period expires, the CONTROL are lowered and the connection is aborted. For a call-in file, the driver will wait for another incoming call; for a call-out file, the *open* will be returned with an EIO error. Once a connection is established, loss of either DSR or CTS will cause the CONTROL to be lowered and, if the device is a controlling terminal, a *hangup* signal will be sent to the controlling process.

If DCD is lost, a timer is started. If DCD resumes before the time period has expired, the connection will be maintained. However, no data transfer will occur during this time. The driver will stop transmitting characters, and any characters received by the driver will be discarded. (However, on some implementations data transmission cannot be stopped. See **DEPENDENCIES**.) If DCD is not restored within the allotted time, the connection will be broken as described above for DSR and CTS.

If the modem connection is to be broken when the *close* system call is issued (i.e. HUPCL is set), then the CONTROL will be lowered and the *close* will be returned as successful. However, no further *open*s will be allowed until after both DSR and CTS have been lowered by the modem, and the hangup timer (see **Modem timers** below) has expired. The action taken in response to an *open* during this time will be the same as if the port were still open. (See **Terminal port access interlock** below.)

When a port is in CCITT mode, the driver has complete control of the modem lines and the user is not allowed to change the setting of the CONTROL or affect which STATUS are actively monitored by the driver (see **Modem ioctls** below). This is to provide strict adherence with the CCITT recommendations.

Simple mode is used for connections to devices which require only a simple method of modem line control. This can include devices such as black boxes, data switches, or for system-to-system connections. It can also be used with modems which cannot operate under the CCITT recommendations. The CONTROL for simple mode consists of only DTR. The STATUS consists of only DCD. When an *open* is issued, the CONTROL is raised but no connection timer is started. When the STATUS becomes raised, a connection is established and the *open* request is returned as SUCCESSFUL. Once a connection is established, loss of the STATUS will cause the CONTROL to be lowered and, if the device is a controlling terminal, a *hangup* signal will be sent to the controlling process.

When a port is in simple mode, the driver will normally control the modem lines. However, the user is allowed to change the setting of the CONTROL (see **Modem ioctls** below).

Terminal port access interlock

An interlock mechanism is provided between the three access types of terminal files. It prevents more than one file from being successfully opened at a time, but allows certain *open*s to succeed while others are PENDING so that a port can be opened through a call-out connection while *getty* has a pending *open* at a call-in connection. The three access types are given a priority that determines which *open* will succeed if more than one file has an *open* issued against it. The three access types are ordered from lowest priority to highest as follows: call-in, call-out, and direct.

If an *open* is issued to a port which already has a SUCCESSFUL *open* on it of a lower priority type, the new *open* will be returned with an EBUSY error. (EBUSY will also be returned by an attempted *open* on a CCITT call-out file if an incoming call indication is currently being received. In this case, if there is a PENDING *open* on the corresponding CCITT call-in file, this PENDING *open* will complete.) If the lower priority *open* is PENDING, the new *open* will succeed if possible, or will be left PENDING if waiting for the STATUS and the lower priority *open* will become BLOCKED. If a higher priority *open* has succeeded or is PENDING, the new *open* will be BLOCKED, unless the new *open* has the O_NDELAY flag bit set, in which case the *open* will be returned with an EBUSY error. Once an *open* on one type of file is SUCCESSFUL, any PENDING *open*s on lower priority files will become BLOCKED.

When a file of one priority is closed, a BLOCKED *open* on the next lower priority type file will become active. If all of the STATUS are raised, the *open* will be SUCCESSFUL, otherwise the *open* will become PENDING waiting for the STATUS. If the lower priority *open* is SUCCESSFUL (because the connection was maintained when the higher priority file was closed), the port characteristics (speed, parity, etc.) that were set by the higher priority file will be inherited by the lower priority file. If the connection is not maintained through the *close*, the port characteristics will be set to default values.

Modem timers

There are four timers currently defined for use with modem connections. The first three of the timers are applicable only to CCITT mode connections. In general, the effect of changing a timer value while the timer is running is system dependent. However, setting the timer value to zero is guaranteed to disable the timer even if it is running.

The connect timer is used to limit the amount of time to wait for a connection to be established once it has been begun. This timer is started when an incoming call has been received on a call-in file, or when an *open* has been issued on a call-out file for which no *open*s are already pending. If the connection is completed in time, the timer is aborted. If the time period expires, the connection is aborted. For a call-in file, the driver will again wait for an incoming call and the *open* will remain pending. For a call-out file, the *open* will be returned with an EIO error.

The carrier detect timer is used to limit the amount of time to wait before causing a disconnect if DCD drops. If carrier is not re-established in this time, a disconnect will occur. If carrier is re-established before the timeout, the timer will be aborted and the connection maintained. During the period when carrier is not raised, no data will be transferred across the line.

The no activity timer is used to limit the amount of time a connection will remain open with no data transfer across the line. When the data line becomes quiescent with no data transfer, this timer will be started. If data is again transferred over the line in either direction before the time limit, the timer will be aborted. If no activity occurs before the timeout has occurred, the driver will disconnect the line. This can be used to avoid long and costly telephone connections when data transfer has been stopped either normally or abnormally.

The last timer defined, the hangup timer, is used for both CCITT and simple modes. This timer controls the amount of time to wait after disconnecting a modem line before allowing another *open*. This time period should be made long enough to guarantee that the connection has been terminated by the telephone switching equipment. If this period is not long enough, the telephone connection may not be broken and a succeeding *open* may complete with the old connection.

Modem ioctls

Several *ioctl* system calls apply to manipulation of modem lines. They use the following information defined in **<sys/modem.h>**:

```
#define NMTIMER     6
typedef unsigned long  mflag;
struct  mtimer  {
        unsigned short m_timers[NMTIMER];
        };
```

Each bit of the *mflag* long corresponds to one of the modem lines as follows:

MRTS	Request to Send	outbound
MCTS	Clear to Send	inbound
MDSR	Data Set Ready	inbound
MDCD	Data Carrier Detect	inbound
MDTR	Data Terminal Ready	outbound
MRI	Ring Indicator	inbound
MDRS	Data Rate Select	outbound

The timer values are defined in the array m_timers. The relative position of the timer and default initial values and units for each timer are as follows:

0	MTCONNECT	25 s
1	MTCARRIER	400 ms
2	MTNOACTIVITY	0 min
3	MTHANGUP	250 ms
4	Reserved	
5	Reserved	

A value of zero for any timer will disable that timer.

The modem line *ioctl* system calls have the form:

int ioctl(int fildes, int command, mflag *arg);

The commands using this form are:

MCGETA Get the current state of both inbound and outbound modem lines and store in the **mflag** long referenced by **arg**. A raised line will be indicated by a one bit in the appropriate position.

MCSETA Set the outbound modem lines from the *mflag* long referenced by **arg**. Setting an outbound bit to one causes that line to be raised and zero to be lowered. Setting bits for inbound lines has no effect. Setting any bits while in CCITT mode has no effect. The change to the modem lines is immediate and using this form while characters are still being output may cause unpredictable results.

MCSETAW Wait for the output to drain and set the new parameters as described above.

MCSETAF Wait for the output to drain, then flush the input queue and set the new parameters as described above.

The timer value *ioctl* system calls have the form:

int ioctl(int fildes, int command, mtimer *arg);

The commands using this form are:

MCGETT Get the current timer value settings and store in the *mtimer* structure referenced by **arg**.

MCSETT Set the timer values from the structure referenced by **arg**.

For any timer, setting the timer value to its previous value has no effect.

WARNING

Occasionally it is possible that a process may open a call-out file at approximately the same time as an incoming call is received. In some cases, the call-out connection may be satisfied by the incoming call. In general, however, the results are indeterminate. If necessary, the situation can be avoided by the use of two modems and ports, one for call-out connections and the other for receiving incoming calls.

DEPENDENCIES

Some hardware implementations may not have access to all modem lines supported by MCSETA. If a

particular hardware does not support a given line, attempts to set the value of a line will be ignored, and reading the current state of the line will return zero. The appropriate I/O card manual should be referenced to determine the lines supported by the hardware installed.

Some hardware implementations may not have access to all timers supported by MCSETT. Also, the granularity of the individual timers may vary depending on the hardware and system in use. The effect of setting a timer out of range or with a granularity outside the capability of a particular system should be documented by that system. The effect of changing the value for a timer while that timer is running is system dependent and should be documented by each system.

Setting the CLOCAL bit while a timer is running will cause the timer to be stopped. It is a system dependency whether or not the timer is restarted, and if so, the value at which it is restarted when the CLOCAL bit is subsequently cleared.

On those implementations supporting the HP27140A 6-Channel Multiplexer, transmission of characters cannot be stopped during loss of DCD. The driver cannot detect loss of DCD until the connection is broken. Also, the I/O card may still have characters in its internal buffers and will still try to transmit them.

FILES

/dev/cua*
/dev/cul*
/dev/tty*
/dev/ttyd*

AUTHOR

modem was developed by HP and AT&T.

SEE ALSO

stty(1), mknod(1M), ioctl(2), open(2), termio(7).

NAME

mt - magnetic tape interface and controls

DESCRIPTION

This entry describes the behavior of HP magnetic tape interfaces and controls, including DDS and QIC cartridge drives. The files `/dev/rmt/*` refer to specific raw tape drives, and the behavior of each given unit is specified in the major and minor numbers of the device special file.

The following naming conventions are recommended because they relate most of the mode flags to the device name:

/dev/rmt/(c#d)#[hml]{c}{n}{b}

or

/dev/rmt/(c#d)#qic[525 | 150 | 120]{n}{b}

In this format, **c#d** indicates the controller number (optionally specified by the system administrator), # is the device number, **hml** indicates the density (**h** (high) for 6250 bpi, **m** (medium) for 1600 bpi, and **l** (low) for 800 bpi), **c** indicates data compression, **n** indicates no rewind on close and **b** indicates Berkeley style. For example, **/dev/rmt/2mn** is device lu 2, AT&T style at 1600 bpi with no rewind and no compression. The selection of controller and unit numbers is system dependent, and is discussed in the appropriate system administrator's manual.

For S800 QIC devices, **qic** (without a format number, i.e. default format) indicates the best capacity format for the drive and currently loaded medium, **qic525** for QIC-525/320 format, **qic150** for QIC-150 format, and **qic120** for QIC-120 format.

Accessing a QIC device through a **/dev/rmt/(c#d)#**[`hml`] device file is equivalent to **/dev/rmt/(c#d)#qic** in that a default format will be selected.

The operation of a tape drive is controlled by mode flags, which are usually encoded as bits in the minor number of the device special file.

no-rewind Unless this mode is requested, the tape is automatically rewound upon close. When a rewind on close is not desired, the **n** flag should be used in the device name.

style When this mode is requested, the tape drive behaves as on Berkeley systems; when not requested, the drive behaves as on AT&T UNIX operating systems. The details are described below. The *ioctl*(2) operations described below work in both modes. The *mt*(1) tape movement utility requires that the Berkeley mode be specified.

density This may be used to select the density of the tape being written. Values that may be selected include 6250, 1600, and 800 bpi, depending on the capabilities of the specific tape drive. This corresponds to the **h, m** and **l** flags in the recommended device name. For DDS (digital audio tape) and QIC (quarter inch tape) format devices, density designations are not used.

format This may be used to select the QIC format of the cartridge being written. See DEPENDENCIES.

compression On tape drives that support data compression, selecting the device file with **c** causes the data to be written or read in compressed mode.

Refer to the system administrator manual for your computer for more specific details of how to select the modes for a given device.

When opened for reading or writing, the tape is assumed to be positioned as desired.

When a file opened for writing is closed, two consecutive EOF marks are written if, and only if, one or more writes to the file have occurred. The tape is rewound unless the no-rewind mode has been specified, in which case the tape is positioned before the second EOF just written. For QIC devices only one EOF mark is written and the tape is positioned after the EOF mark if the no-rewind mode has been specified.

When a file open for reading only is closed and the no-rewind bit is not set, the tape is rewound. If the no-rewind bit is set, the behavior depends on the *style* mode. For AT&T-style devices, the tape is positioned after the EOF following the data just read. For Berkeley-style devices, the tape is not repositioned in any way.

Each *read*(2) or *write*(2) call reads or writes the next record on the tape. For writes, the record has the same length as the buffer given (within the limits of the hardware).

During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. The number of bytes ignored (for records longer than the buffer size specified) is available in the **mt_resid** field of the **mtget** structure via the MTIOCGET call of *ioctl*(2). The buffer and size might have implementation-dependent alignment restrictions.

Reading an EOF mark is returned as a successful zero-length read; that is, the data count returned is zero and the tape is positioned after the EOF, enabling the next read to return the next record.

DDS format devices also support **setmarks** which are hierarchically superior to **filemarks**. A setmark is used to delineate a group (set) of files. Reading a setmark is also returned as a zero-length read. The two can be distinguished by unique bits in the **mt_gstat** field.

Spacing operations (back or forward space, setmark, file or record) leave the tape positioned past the object being spaced to in the direction of motion. In other words, backspacing a file leaves the the tape positioned before the file mark; forward spacing a file leaves the tape positioned after the file mark. This is consistent with all classical usage on tapes.

Seeks on a magnetic tape device are ignored. Instead, the *ioctl*(2) operations below can be used to position the tape and determine its status.

The header file <sys/mtio.h> has useful information for tape handling. The following is included from **<sys/mtio.h>** and describes the possible tape operations:

```
/* mag tape I/O control requests */

#define MTIOCTOP    _IOW(m,1,struct mtop)    /* do mag tape op */
#define MTIOCGET    _IOR(m,2,struct mtget)   /* get tape status */

/* structure for MTIOCTOP - mag tape op request */

struct          mtop {
        short    mt_op;      /* operations defined below */
        daddr_t  mt_count;   /* how many of them */
};

/* operations */

#define MTWEOF    0    /* write filemark (end-of-file record) */
#define MTFSF     1    /* forward space file */
#define MTBSF     2    /* backward space file */
#define MTFSR     3    /* forward space record */
#define MTBSR     4    /* backward space record */
#define MTREW     5    /* rewind */
#define MTOFFL    6    /* rewind, put drive offline */
#define MTNOP     7    /* no-op, may set status */
#define MTEOD     8    /* DDS and QIC only. seek to end-of-data */
#define MTWSS     9    /* DDS only. write setmark(s) */
#define MTFSS     10   /* DDS only. space forward setmark(s)*/
#define MTBSS     11   /* DDS only. space backward setmark(s)*/

/* structure for MTIOCGET - mag tape get status command */

struct          mtget {
        long    mt_type;
        long    mt_resid;

/* The following two registers are device dependent */

        long    mt_dsreg1;
        long    mt_dsreg2;

/* The following is a device-independent status word */

        long    mt_gstat;
        long    mt_erreg;
```

Information for decoding the mt_type field can be found in <sys/mtio.h>.

EXAMPLES

Assume that *fd* is a valid file descriptor. The first example writes two consecutive filemarks on the tape:

```
#include <sys/types.h>
#include <sys/mtio.h>

struct  mtop mtop;

mtop.mt_op = MTWEOF;
mtop.mt_count = 2;
ioctl(fd, MTIOCTOP, &mtop);
```

If *fd* is a valid file descriptor for an open DDS drive, the following example spaces forward to just past the next setmark:

```
#include <sys/types.h>
#include <sys/mtio.h>

struct  mtop mtop;

mtop.mt_op = MTFSS;
mtop.mt_count = 1;
ioctl(fd, MTIOCTOP, &mtop);
```

Now suppose that *fd* is a valid file descriptor for an opened tape device, and suppose further that it has just returned zero from a *read*(2) request. To verify that the tape has just read a filemark, the application could issue the following system call:

```
#include <sys/types.h>
#include <sys/mtio.h>

struct mtget mtget;

ioctl(fd, MTIOCGET, &mtget);
if (GMT_EOF(mtget.mt_gstat)) {
/* code for filemark detection */
}
```

WARNINGS

It is impossible to write a program that leaves a tape positioned at the beginning of the tape on an AT&T-style device with the no-rewind bit set because closing the device file upon the program's termination repositions the tape after the first EOF mark.

An AT&T-style device file opened for writing to blank media may cause an error condition at close (due to attempting to space to the non-existent next EOF mark) unless a tape alteration operation has been performed.

HP-UX silently enforces a tape record blocking factor (MAXPHYS) on large I/O requests. For example, a user write request with a length of ten times MAXPHYS will actually reach the media as ten separate records. A subsequent read (with ten times MAXPHYS as a length) will look like a single operation to the user, even though HP-UX has broken it up into ten separate read requests to the driver. Such activity is normally transparent to the user unless:

- The user picks an arbitrary read length that is greater than MAXPHYS.
- The user attempts to read a third-party tape containing records larger than MAXPHYS.

Since the value for MAXPHYS is relatively large (usually >= 64K bytes), this is typically not a problem.

Write operations on a QIC device can be initiated only at BOT or EOD. No overwriting is allowed by positioning the tape in the middle of recorded data.

The offline operation puts the QIC drive offline. The cartridge is not ejected as done for DDS. To put the drive back online, the cartridge has to be manually ejected and then reinserted.

Sequential-Access devices that use the SCSI-1 I/O interface do not always report true media position.

DEPENDENCIES

Series 300/400

QIC is not supported.

Series 800

The MTNOP operation does not set the device-independent status word.

QIC devices do not always report media position accurately.

If no QIC specific format is specified in the minor number, the best capacity format for the drive and currently loaded medium is used.

The maximum I/O request for QIC devices is limited to 64K - 1 (65535) bytes.

Efficient use of streaming tape drives with large internal buffers and immediate-reporting require the following end-of-tape procedures:

All writes near the EOT foil (which is not on the recording surface) complete without error if actually written to the tape. Once the tape drive determines that the foil has been passed, subsequent writes do not occur and an error message is returned.

Since some applications require that a trailer be written for multiple tape operations, a user request for magnetic tape status that reflects the EOT condition signals the driver to drop all write barriers. Caution must be exercised to keep the tape on the reel.

When reading near the end-of-tape, the user is not informed of the EOT foil marker. Instead, the typical double EOF marks or a pre-arranged trailer signals the logical end-of-tape.

The EOT description above applies in the default case when immediate-reporting mode is allowed by a value encoded in the minor number. When not permitted by the minor number, the EOT operation attempts to emulate compatibility-mode on other HP-UX machines. In this mode, the write encountering the EOT foil returns an error with the tape automatically backing up over that record. The read encountering the EOT foil returns an error.

Since magnetic tape drives vary in EOT sensing due to differences in the physical placement of sensors, any application (such as multiple-tape *cpio*(1) backups) requiring that data be continued from the EOT area of one tape to another tape must be restricted. Therefore, the tape drive type and mode should be identical for the creation and reading of the tapes.

The following macros are defined in **<sys/mtio.h>** for decoding the generic status of the tape drive (returned in the **mt_gstat** field):

```
GMT_BOT(x)          /* At beginning of tape */
GMT_EOD(x)          /* DDS and QIC End-of-Data encountered */
GMT_EOF(x)          /* At an EOF mark */
GMT_EOT(x)          /* At end of tape */
GMT_DR_OPEN(x)      /* Drive door is open */
GMT_IM_REP_EN(x)    /* Immediate reporting mode enabled */
GMT_ONLINE(x)       /* Drive is on line */
GMT_SM(x)           /* setmark encountered */
GMT_WR_PROT(x)      /* Tape is write protected */
GMT_D_6250(x)       /* Density is 6520 bpi */
GMT_D_1600(x)       /* Density is 1600 bpi */
GMT_D_800(x)        /* Density is 800 bpi */
GMT_COMPRESS(x)     /* Data compression enabled */
GMT_QIC_FORMAT(x)   /* QIC format on tape */
GMT_QIC_MEDIUM(x)   /* QIC medium type*/
```

If GMT_IM_REP_EN(x) is true, the drive reports completion of each operation immediately after receiving it.

AUTHOR

mt was developed by HP and the University of California, Berkeley.

FILES

/dev/rmt/*

SEE ALSO
dd(1), mt(1), ioctl(2), ct(7).

NAME

nfs, NFS - network file system

DESCRIPTION

The Network File System (NFS) allows a client node to perform transparent file access over the network. By using NFS, a client node operates on files residing on a variety of servers and server architectures, and across a variety of operating systems. File access calls on the client (such as read requests) are converted to NFS protocol requests and sent to the server system over the network. The server receives the request, performs the actual file system operation, and sends a response back to the client.

NFS operates in a stateless manner using remote procedure calls (RPC) built on top of an external data representation (XDR) protocol. The RPC protocol enables version and authentication parameters to be exchanged for security over the network.

A server grants access to a specific file system to clients by adding an entry for that file system to the server's **/etc/exports** file.

A client gains access to that file system using the **mount** command to request a file handle for the file system (see *mount*(1M)). (A file handle is the means by which NFS identifies remote files.) Once a client mounts the file system, the server issues a file handle to the client for each file (or directory) the client accesses. If the file is removed on the server side, the file handle becomes stale (dissociated with a known file), and the server returns an error with **errno** set to ESTALE.

A server can also be a client with respect to file systems it has mounted over the network; however, its clients cannot directly access those file systems. If a client attempts to mount a file system for which the server is an NFS client, the server returns with **errno** set to EREMOTE. The client must mount the file system directly from the server on which the file system resides.

The user ID and group ID mappings must be the same between client and server. However, the server maps UID 0 (the super-user) to UID –2 before performing access checks for a client. This process prevents gaining super-user privileges on remote file systems.

RETURN VALUE

Generally, physical disk I/O errors detected at the server are returned to the client for action. If the server is down or inaccessible, the client receives the message:

```
NFS:  file server not responding: still trying.
```

The client continues resending the request until it receives an acknowledgement from the server. Therefore, the server can crash or power down, and come back up without any special action required by the client. The client process requesting the I/O will block, but remains sensitive to signals (unless mounted with the **ointr** option) until the server recovers. However, if mounted with the **soft** option, the client process returns an error instead of waiting indefinitely.

AUTHOR

nfs was developed by Sun Microsystems, Inc.

SEE ALSO

mount(1M), nfsd(1M), vfsmount(2), checklist(4), exports(4).

NAME
null - null file

DESCRIPTION
Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

EXAMPLES
To create a zero-length file, use either of the following:

cat /dev/null >*file*
cp /dev/null *file*

FILES
`/dev/null`

STANDARDS CONFORMANCE
`null`: AES, SVID2, XPG2, XPG3, XPG4, FIPS 151-2, POSIX.1, POSIX.2

NAME

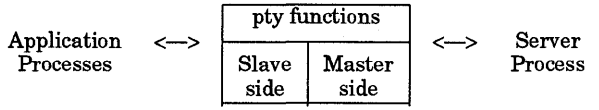
pty - pseudo terminal driver

SYNOPSIS

```
pseudo-device pty
```

DESCRIPTION

The `pty` driver provides support for a device-pair termed a pseudo terminal. A pseudo terminal is a pair of character devices, a master device and a slave device. The slave device provides to application processes an interface identical to that described in *termio*(7). Unlike all other devices that provide the interface described in *termio*(7), the slave device does not have a hardware device behind it. Instead, it has another process manipulating it through the master half of the pseudo terminal. Thus anything written on the master device is given to the slave device as input, and anything written on the slave device is presented as input on the master device.



Open and Close Processing

The slave side of the pty interprets opening or closing the master side as a modem connection or disconnection on a real terminal. Only one open to the master side of a pty is permitted. An attempt to open an already open master side returns `-1` and sets the external variable `errno` to EBUSY. An attempt to open the master side of a pty that has a slave with an open file descriptor returns `-1` and sets `errno` to EBUSY. The potential problem of ptys being found busy at opens can be avoided by using the *clone open* functionality discussed in the next section.

An attempt to open a non-existent pty returns `-1` and sets `errno` to ENXIO. If `O_NDELAY` is not specified, opens on the slave side hang until the master side is opened. If `O_NDELAY` is specified, opens on the slave side return error if the master side is closed. Any `ioctl()` (see *ioctl*(2)) or `write()` (see *write*(2)) request made on the slave side of a pty after the master side is closed returns `-1` and sets the external variable `errno` to EIO. A `read()` request made on the slave side of a pty after the master side is closed returns 0 bytes (see *read*(2)). Closing the master side of a pty sends a `SIGHUP` hangup signal to the tty process group number of the corresponding slave side and flushes pending input and output.

Clone Open

In typical pty usage, there is no preference among pty pairs. Thus, it is useful to be able to issue a single `open()` that internally opens any available pty (see *open*(2)). An open on `/dev/ptym/clone` returns an open file descriptor of a free master pty device. If there are no free devices, the open returns `-1` and sets `errno` to EBUSY. The name of the slave device corresponding to the opened master device can be found through a `ptsname()` request (see *ptsname*(3C)).

Processing ioctl() Requests

By default, any `ioctl()` request defined by *termio*(7) is recognized by both the master and slave sides of a pty. These `ioctl()` requests are processed by the pty driver as specified by *termio*(7). In addition, the `ioctl()` requests defined below are recognized by the master side of a pty. The slave side only recognizes `ioctl()` requests defined by *termio*(7). An `ioctl()` request made on the slave side of a pty after the master side is closed returns `-1` and sets the external variable `errno` to EIO. An `ioctl()` request not recognized by the pty returns `-1` and sets the external variable `errno` to EINVAL. Note that some of the master-side-only `ioctl()` requests affect which `ioctl()` requests are recognized by the master and slave side of the pty. These master-side-only `ioctl()` requests also affect the way recognized `ioctl()` requests, `open()` requests, and `close()` requests are processed by the pty driver (see *close*(2)).

The following `ioctl()` requests, defined in <`sys/ptyio.h`>, apply only to the master side of pty:

`TIOCSIGSEND`

Cause a signal to be sent from the slave side of the pty to the current tty process group of the slave side. The value of the parameter is taken to be the signal number sent. An EINVAL error is returned and no signal is sent if the specified signal number does not refer to a legitimate signal (see *signal*(5)). Note that this request allows the server process to send signals to processes not owned by the same user ID.

TIOCTTY Enable or disable all termio processing by a pty. termio processing is enabled if the int addressed by *arg* is nonzero and disabled if the int addressed by *arg* is zero. By default, termio processing is enabled. termio processing refers to processing of input and output described by *termio*(7) (such as tab expansion), as well as the processing of the ioctl() requests described by *termio*(7). When disabled, all input and output data is passed through the pty without modification. Issuing a TIOCTTY ioctl() request flushes all data buffered in the pseudo terminal and releases any processes blocked waiting for data. Enabling and disabling TIOCTTY affects the operation of the following ioctl() requests: TIOCPKT, TIOCREMOTE, TIOCBREAK, TIOCSTOP, TIOCSTART, TIOCTRAP, and TIOCMONITOR.

When TIOCTTY is enabled, all termio ioctl() requests execute as specified in *termio*(7), regardless of the side from which the ioctl() request is made. When TIOCTTY is disabled, master side termio ioctl() requests set and return the the external variable errno to EINVAL. Slave side termio ioctl() requests are processed like any other ioctl() request when TIOCTTY is disabled. In particular, slave side termio ioctl() requests set and return the external variable errno to EINVAL when both TIOCTTY and TIOCTRAP are disabled. (See the discussion of ioctl(), open(), close() trapping below). ioctl() requests not defined by *termio*(7) are not affected by the state of TIOCTTY.

Data written through a pseudo terminal with TIOCTTY disabled is handled in a manner similar to data flowing through a pipe. A write request blocks in the pty until all data has been written into the pty. A read request blocks if there is no data available unless the O_NDELAY flag is set (see *fcntl*(2)). When data is available to be read, the read request returns whatever is available, and does not wait for the number of bytes requested to be satisfied. The number of bytes a pty can contain in its internal memory is implementation dependent, but is at least 256 bytes in each direction. For example, a write on the slave side of a pty of 1024 bytes might be read on the master side by four read requests returning 256 bytes each. The size of the chunks of data that are read is not guaranteed to be consistent, but no data is lost.

The following ioctl() requests, defined in <sys/ptyio.h>, apply only to the master side of a pty. In particular, these ioctl() requests enable/disable specific modes of pty driver operation. These ioctl() requests work in series with TIOCTTY; that is, the mode must be enabled by its ioctl() request and TIOCTTY must be enabled for the mode to operate. The mode can be enabled or disabled regardless of the state of TIOCTTY.

TIOCPKT Enable or disable packet mode. Packet mode is enabled if the int addressed by *arg* is nonzero and disabled if the int addressed by *arg* is zero. By default, packet mode is disabled. When applied to the master side of a pseudo terminal, each subsequent read() from the master side returns data written on the slave part of the pseudo terminal preceded by a zero byte (symbolically defined as TIOCPKT_DATA), or a single byte reflecting control status information. The value of such a status byte is composed of zero or more bit flags:

TIOCPKT_FLUSHREAD
The read queue for the slave side has been flushed.

TIOCPKT_FLUSHWRITE
The write queue for the slave side has been flushed.

TIOCPKT_STOP
Data flowing from the slave side of the pty to the master side has been stopped by means of ^S, TIOCSTOP, or TCXONC.

TIOCPKT_START
Data flowing from the slave side of the pty to the master side has been restarted.

TIOCPKT_DOSTOP
Stop and start characters have been set to ^S or ^Q.

TIOCPKT_NOSTOP
Stop and start characters are set to something other than ^S or ^Q.

TIOCREMOTE Enable or disable remote mode. Remote mode is enabled if the **int** value of *arg* is nonzero and disabled if the **int** value of *arg* is zero. By default, remote mode is disabled. Remote mode is independent of packet mode. This mode causes input to the pseudo terminal to be flow controlled and not input edited (regardless of the terminal mode). Each write to the master side produces a record boundary for the process reading the slave side. In normal usage, writing data is like typing the data as a line on a terminal; writing zero bytes is equivalent to typing an end-of-file character (that is, the EOF character as defined in *termio*(7)). The data read by the slave side is identical to the data written on the master side. Data written on the slave side and read on the master side with **TIOCREMOTE** enabled is still subject to the normal *termio*(7) processing. **TIOCREMOTE** can be used when doing remote line editing in a window manager, or whenever flow-controlled input is required. Issuing a **TIOCMONITOR** **ioctl()** request flushes all data buffered in the pseudo terminal.

The following **ioctl()** requests, defined in <**sys/ptyio.h**>, apply only to the master side of pty. In particular, these **ioctl()** requests are only recognized when **TIOCTTY** is enabled. When **TIOCTTY** is disabled, these **ioctl()** requests set and return the external variable **errno** to EINVAL.

TIOCBREAK Cause a break operation to be done on the slave side of the pty, as if a user had pressed the break key on a real terminal. Takes no parameter.

TIOCSTOP Stop data flowing from the slave side of the pty to the master side (equivalent to typing **^S**). Takes no parameter.

TIOCSTART Restart output (stopped by **TIOCSTOP** or by typing **^S**). Takes no parameter.

Flow-Control Input and Output Processing

The following terms are used to describe the flow of data through psuedo terminals. INPUT refers to data flowing from the master side of a pty to the slave side. OUTPUT refers to data flowing from the slave side of a pty to the master side.

When packet mode (**TIOCPKT**) is disabled and INPUT is stopped (see IXOFF, input modes, in *termio*(7)), the next **read()** from the master side of a pty returns a STOP character. When INPUT is restarted, the next **read()** from the master side returns a START character. If packet mode (**TIOCPKT**) is enabled, the STOP or START character is preceded by a data packet indicator (**TIOCPKTDATA**). **select()** should be used by the master-side server before each **write()** request to properly handle INPUT flow control (see *select*(2)).

When INPUT flow control is enabled, **write()** and **select()** are handled as follows: Write-selects on the master side of a pty return true only if INPUT has not been stopped. If INPUT becomes stopped while data is being written into the master side of a pty, the write returns with the number of bytes written before INPUT was stopped. Writes done after INPUT is stopped return immediately with zero bytes written.

When packet mode (**TIOCPKT**) is disabled and OUTPUT is stopped (see IXON, input modes in *termio(7)),* each subsequent **read()** from the master side of a pty returns with no data read. When OUTPUT is restarted, each subsequent **read()** from the master side returns data written on the slave side. If packet mode (**TIOCPKT**) is enabled, the first **read()** after OUTPUT has been stopped returns a **TIOCPKTSTOP** packet. All subsequent reads from the master side while OUTPUT is stopped returns a **TIOCPKTDATA** packet with no data. When OUTPUT is restarted, the next **read()** from the master side returns a **TIOCPKTSTART** packet. All subsequent reads from the master side return data written on the slave side preceded by a **TIOCPKTDATA** packet. **select()** should be used by the master-side server before each **read()** to properly handle OUTPUT flow control. Otherwise, reads from the master side of a pty will not be prevented when OUTPUT is stopped.

Trapping ioctl(), open(), close() Requests

When trapping is enabled, the master side is notified when the application on its slave side makes an **ioctl()**, **open()**, or **close()** request. For trapped **ioctl()** and **open()** requests, the slave side is blocked (that is, the request does not complete) until the server on its master side acknowledges the trapped request. For trapped **close()** requests, the slave slave does not block for an acknowledgement.

select() should be used by the master side server to receive notification of trapped **ioctl()**, **open()**, and **close()** requests. When one of these requests is trapped, the **select()** returns with an "exceptional condition" indicated for the slave side's file descriptor. Other mechanisms for receiving notification of trapped requests are defined below, but these mechanisms should be used only if **select()** is not

available.

When trapping is disabled (default condition), `ioctl()` requests made to the slave side set fail and return the external variable `errno` to EINVAL if they are not recognized by the slave side of the pty driver. The only `ioctl()` requests recognized by the slave side are those defined by *termio*(7) and only when `TIOCTTY` is enabled. When `TIOCTTY` is disabled, no `ioctl()` requests are recognized by the slave side. If trapping is enabled and the master side closes, trapping is disabled. If the master closes during the middle of a handshake with the slave, the handshake is done automatically.

Trapping occurs in two forms that are identified by the `ioctl()` requests that enable or disable them — `TIOCTRAP` and `TIOCMONITOR`. These two forms are distinguished by the types of requests they affect and by the capabilities they provide. Trapping `open()` and `close()` requests is enabled or disabled by `TIOCTRAP`. Trapping `ioctl()` requests not defined by *termio*(7) are enabled or disabled by `TIOCTRAP`. Trapping `ioctl()` requests defined by *termio*(7) are enabled or disabled by `TIOCTRAP` only when `TIOCTTY` is also disabled. When `TIOCTTY` is enabled, trapping `ioctl()` requests defined by *termio*(7) are enabled or disabled by `TIOCMONITOR`. Briefly, both `TIOCTRAP` and `TIOCMONITOR` trapping allow the server on the master side to examine the request's parameters, the pid making the request, etc. In addition, `TIOCTRAP` trapping allows the server to modify the parameters and return values of an `ioctl()` request.

The following `ioctl()` calls apply only to the master side of a pty and pertain to trapping `ioctl()`, `open()`, and `close()` requests. They are defined in <`sys/ptyio.h`>:

`TIOCTRAP` Enable or disable trapping of `ioctl()`, `open()`, and `close()` requests made by the application on the slave side of a pty. Trapping is enabled if the `int` addressed by *arg* is nonzero and disabled if the `int` addressed by *arg* is zero. By default, `TIOCTRAP` trapping is disabled.

`TIOCTRAPSTATUS`
Check for a pending `ioctl()`, `open()`, or `close()` trap. The argument points to an `int` that is set to one if a trap is pending and to zero if nothing is pending. Use `TIOCTRAPSTATUS` when the preferred method of a `select()` "exceptional condition" is not available.

`TIOCREQCHECK`
Return the trapped `ioctl()`, `open()`, or `close()` information to the master side. Use `TIOCREQCHECK` in response to either a `select()` "exceptional condition" or a `TIOCTRAPSTATUS` indicating that a trap is pending. A `TIOCREQCHECK` reads the pending `ioctl()`, `open()`, or `close()` information into the memory pointed to by the *arg* of `TIOCREQCHECK`. The information takes the form of the following `request_info` structure, defined in <`sys/ptyio.h`>:

```
struct request_info {
    int request;
    int argget;
    int argset;
    short pgrp;
    short pid;
    int errno_error;
    int return_value;
};
```

All elements of `request_info` refer to the slave side of the pty and include the following:

`request` The `ioctl()` command received.

`argget` The `ioctl()` request applied to master side to receive the trapped `ioctl()` structure, if one exists (a zero value means there is none). (When nonzero, `argget` is a `TIOCARGGET` request with the size field precomputed.)

`argset` The `ioctl()` request applied to master side to send back the resulting `ioctl()` structure, if one exists (a zero value means there is none). (When nonzero, `argset` is a `TIOCARGSET` request with

the size field precomputed.)

`pgrp` The process group number of the process doing the operation.

`pid` The process ID of the process doing the operation.

`errno_error`
The `errno` external variable error code (initialized to zero) returned by `ioctl()` on the slave side.

`return_value`
The success value (initialized to zero) returned by `ioctl()` on the slave side when `errno_error` is not set.

When the `ioctl()` argument received on the slave side is not a pointer, its value is stored as four bytes retrievable with an `ioctl()` request to the master side equal to `argget`.

When an `open()` or `close()` is being passed, `request` is set to `TIOCOPEN` or `TIOCCLOSE`, respectively. For `TIOCOPEN` and `TIOCCLOSE`, both `argget` and `argset` are zero because there is no `ioctl()` structure. When `TIOCTTY` is enabled, the *termio*(7) definition of open/close is executed first before being passed to the master side. Note that while all opens are trapped, only the last close on a particular inode for a pty slave side is trapped by the pty.

A `TIOCREQCHECK` returns the external variable `errno` error EINVAL if no `ioctl()`, `open()`, or `close()` trap is pending. Accordingly, a `TIOCREQCHECK` that returns EINVAL in response to a `select()` "exceptional condition" indicates that the trapped `ioctl()`, `open()`, or `close()` request was terminated by a signal after `select()` returned.

`TIOCREQGET` Identical to `TIOCREQCHECK` except when no `ioctl()`, `open()`, or `close()` trap is pending. A `TIOCREQGET` blocks until a slave side `ioctl()`, `open()`, or `close()` is trapped; whereas a `TIOCREQCHECK` returns EINVAL. Use `TIOCREQGET` when neither the preferred method of a `select (2)` "exceptional condition" nor the master side `ioctl()` `TIOCTRAPSTATUS` is available.

`TIOCREQSET` Complete the handshake started by a previous `TIOCREQCHECK` or `TIOCREQGET`. The argument should point to the `request_info` structure, as defined by the `TIOCREQCHECK`.

Before doing this `ioctl()` request to complete the handshake, the server should set `errno_error` to an external variable `errno` error value to be passed back to the slave side. If there is no error, `errno_error` can be left alone because the pty initializes it to zero. Also, when there is no error, `return_value` should be set if other than a zero result is desired. The server can set `return_value` and `errno_error` if the trapped request is an `ioctl()`. Setting either `return_value` or `errno_error` for a trapped `open()` or `close()` affects neither the return value of the request nor the external variable `errno` value of the slave side. Further, setting either `return_value` or `errno_error` does not cause `TIOCREQSET` to return an error to the server.

If the `TIOCREQSET` request is made and the request value in the passed `request_info` structure does not equal the trapped value, the external variable `errno` is set and returned as EINVAL. EINVAL is also returned if there are no trapped `ioctl()`, `open()`, or `close()` requests. If the trapped request has been interrupted by a signal between the time that the server has done the `TIOCREQGET` and the `TIOCREQSET`, the `TIOCREQSET` request returns EINVAL.

`TIOCMONITOR`
Enable or disable read-only trapping of `termio ioctl()` requests. `TIOCMONITOR` trapping is enabled if the `int` addressed by *arg* is nonzero and disabled if the `int` addressed by *arg* is zero. By default, `TIOCMONITOR` trapping is disabled. `TIOCMONITOR` works in series with `TIOCTTY`; that is, the `TIOCMONITOR` trapping must be enabled and `TIOCTTY` must be enabled for `termio ioctl()` requests to be trapped by `TIOCMONITOR`. `TIOCMONITOR` trapping can be enabled or disabled regardless of the state of

TIOCTTY.

When TIOCTTY is disabled, termio ioctl() requests are not trapped by TIOCMONITOR. However, ioctl() requests are trapped by TIOCTRAP if TIOCTTY is disabled and TIOCTRAP is enabled. TIOCTRAP trapping allows the master side server to modify the parameters and return values of an ioctl() request, whereas TIOCTMONITOR trapping does not.

TIOCMONITOR trapping allows the server on the master side to know when characteristics of the line discipline in the pty are changed by an application on its slave side. The mechanism for handshaking termio requests trapped by TIOCMONITOR is the same as the mechanism described above for requests trapped by TIOCTRAP. (It is recommended that termio ioctl() requests be used on the master side to interrogate the configured state of the line discipline in the pty. This compensates for the window of time before TIOCMONITOR is enabled, when termio ioctl() requests are not trapped.)

When using select() on the master side of a pty, the "exceptional condition" refers to an open(), close(), or ioctl() request pending on the slave side, while "ready for reading or writing" refers to a read() or write() request pending.

Of the ioctl() requests subject to being trapped, only one-per-pty can be handled at a time. This means that when an application does a non-termio ioctl() request to the slave side, all other ioctl() requests to the same pty slave side are blocked until the first one is handshaked back by the master side. (ioctl() requests that are not trapped, such as termio when TIOCTTY is enabled and TIOCMONITOR is disabled, are not blocked.) This permits the implementation of indivisible operations by an ioctl() call on the slave side that is passed to the server process.

In summary, the following method of handling trapped ioctl(), open(), and close() requests is preferred:

1. Call select(). This system call blocks the master side until a slave side ioctl(), open(), or close() request is trapped.
2. Make TIOCREQCHECK ioctl() request. This step returns information about a trapped ioctl(), open(), or close() request. If TIOCREQCHECK returnsthe external variable errno error EINVAL, loop back to the select() call.
3. Make argget ioctl() request. This optional step is used if argget is nonzero and the server wants to do more than just reject the trapped slave ioctl() request.
4. Make argset ioctl() request. This optional step is done if argset is nonzero and the server wants to pass back a modified ioctl() structure. It is done after the trapped ioctl() request is processed via the server on the master side.
5. Set errno_error and return_error. If the trapped request is an ioctl(), set errno_error appropriately. If the appropriate value for errno_error is zero, return_error must be set.
6. Make TIOCREQSET ioctl() request. This step completes the trapped ioctl(), open(), close() request.

While a process is waiting in the slave side of the pty for the server to complete a handshake, it is susceptible to receiving signals. The following master side ioctl() request allows the server process to control how the pty responds when a signal attempts to interrupt a trapped open() or ioctl() request:

TIOCSIGMODE

Set the signal handling state of the pty to the mode specified as the argument. The mode can have three values, which are TIOCSIGBLOCK, TIOCSIGABORT, and TIOCSIGNORMAL.

TIOCSIGBLOCK

Cause some signals to be postponed that are destined for the slave-side process whose open() or ioctl() request is trapped. Signals are postponed if they would otherwise cause the process to jump to an installed signal handler. Signals are not postponed if they would otherwise cause the process to abort or if they are being ignored. When the server process completes the handshake by

means of the TIOCREQSET ioctl() request, the process returns to the calling program and any pending signals are then acted upon. Any signals that the user has blocked by means of sigblock() continues to be blocked.

TIOCSIGABORT
Prevent a trapped open() or ioctl() request from being restarted. The server process sets this mode when it wants the interrupted requests to return to the calling program with an EINTR error.

TIOCSIGNORMAL
This is the default mode of the pty. If a signal interrupts a trapped open() or ioctl() request, the user's signal handler routine can specify whether the request is to be restarted. If the request is restarted, it executes again from the beginning and the server has to make another TIOCREQGET request to start the handshake over again. If the user's signal handler routine specifies that the interrupted request should not be restarted, the request returns to the calling program with EINTR upon completion of the signal handler. Note that the restarted request is not necessarily the very next one to be trapped.

WARNINGS

The slave side cannot indicate an end-of-file condition to the master side.

When using TIOCREMOTE, a single write() request to the master side of greater than 256 bytes may result in multiple smaller records being read from the slave side instead of only one record.

DEPENDENCIES

Series 300/400

The *clone open* functionality is currently supported on the Series 800 systems only.

The largest ioctl() argument passable between master and slave sides is currently limited to 128 bytes.

Slave-side non-termio ioctl() requests either go unrecognized or are passed to the master side, depending upon the state of the TIOCTRAP.

Series 700

The *clone open* functionality is currently supported on the Series 800 systems only.

AUTHOR

pty was developed by the University of California, Berkeley.

FILES

/dev/ptym/pty[a-ce-z][0-9][0-9]	master pseudo terminals
/dev/ptym/pty[a-ce-z][0-9a-f]	master pseudo terminals
/dev/pty[pqr][0-9a-f]	master pseudo terminals
/dev/pty/tty[a-ce-z][0-9][0-9]	slave pseudo terminals
/dev/pty/tty[a-ce-z][0-9a-f]	slave pseudo terminals
/dev/tty[pqr][0-9a-f]	slave pseudo terminals

SEE ALSO

ioctl(2), select(2), signal(5), termio(7).

NAME

routing - system support for local network packet routing

DESCRIPTION

The network facilities provide general packet routing, leaving most routing table maintenance to applications processes.

A simple set of data structures comprise a **routing table** used in selecting the appropriate remote host or gateway when transmitting packets. The table contains a single entry for each route to a specific network or host.

The table contains the entry `lo0` for the local loopback after system boot-up, and an entry (`lan0`, `lan1`, `lan2`,...) for each interface card after the `ifconfig` command is executed (see *ifconfig*(1m)).

The super-user can change the table by using the *route*(1m) command (see *route*(1M), or by information received in Internet Control Message Protocol (ICMP) redirect messages.

To display the routing table, use the `netstat -r` command (see *netstat*(1)) which displays the destination internet address, which gateway to use to get to that destination, and flags. A routing table contains three types of entries: entries for a specific host, entries for all hosts on a specific network, and entries for any destination not matched by entries of the first two types (a wildcard route). The various types of routes are determined by the `flags` field of the display from `netstat`. The flags field is either `U`, `UG`, `UH`, or `UGH`. The `U` flag is always present. The `G` flag indicates a route using a gateway and is accompanied by a hop count. If a route is via a remote gateway, the hop count must be greater than zero. If no `G` flag is present, it indicates a route that does not use a remote gateway. The `H` flag indicates a route to a host. If the `H` flag is absent, it indicates a route to a network.

The keyword, `default`, in the destination field indicates a wildcard gateway. This is used as a last resort if no information exists in the table about how to get to a particular remote network. Routes that are not valid are not displayed (see Flags below for details).

First, an attempt is made to find a route to the specific host. If that search fails, the algorithm looks for a route to the host's network. If both searches fail, the wildcard gateway is returned if one is in the table. If there are multiple routes of the same type (in other words, two routes to a host via different gateways, two routes to a network via different gateways, or two default routes), the packet is routed over the first route of the same type found in the route table (i.e, the first route of the same type displayed by `netstat -r`).

If all of the searches fail, an error is returned.

Flags

The following truth table shows the relationship between the *count* parameter used with the `route` command and the destination type, flags, and route type.

Count	Destination Type	Flags	Route Type
=0	network	1=U	route to a network via a gateway which is the local host itself
>0	network	3=UG	route to a network via a gateway which is a remote host
=0	host	5=UH	route to a host via a gateway which is the local host itself
>0	host	7=UGH	route to a host via a gateway which is a remote host
=0	"default"	1=U	wildcard route via the local host
>0	"default"	3=UG	wildcard route via a remote gateway

The routing algorithm used includes the ability to recognize **subnets**. Subnet addresses are similar to the network address portion of Internet addresses; network addresses identify physically distinct networks; subnet addresses identify physically distinct subnetworks of the same network. Subnets allow a network manager to partition the host number space associated with a given network number into discrete subnetworks. This facility is desirable if it is necessary for several physical networks to share a single network number. An example is a facility with a single class B network number and several Ethernet-like physical networks. The host space of a class B address is 16 bits, while a single physical network can have a

limitation of 200-300 hosts. If subnets are used it is possible for all of the networks to have the same network number while each host recognizes that another host with the same network number is not necessarily on the same physical network. The routing algorithm attempts to find a gateway for a host if it is not on the same subnet, although it may have the same network number.

The subnet for a given host is specified in the `ifconfig` command (see *ifconfig*(1M)). It is specified as a 32-bit subnet *mask*. The next paragraph outlines an example use of subnets.

An example class C network number is 192.34.17.0, with the last field specifying the host number. Normally all hosts with the prefix 192.34.17 are recognized as being on the same logical and physical network. If subnets are not in use, the default mask used is 255.255.255.0. When routing, bit-wise logical ANDs are performed between the mask and the Internet address of the remote host, and between the mask and the local Internet address. If the result is non-zero, it is assumed that the remote host is on the same subnet as the local host. If subnets are to be used and the 8-bit host field is to be partitioned into 2 bits of subnet and 6 bits of host, the subnet mask would be 255.255.255.192. Note that a subnet mask of all zeroes causes the routing mechanism to assume that all hosts, whether local or remote, are on the local physical network.

If a subnet mask is not specified in the `ifconfig` command, the default mask is used to indicate that subnets are not in use. The default masks for the various classes of Internet addresses are as follows:

Class A: 255.0.0.0
Class B: 255.255.0.0
Class C: 255.255.255.0

WARNINGS

Reciprocal *route* commands must be executed on the local host and the destination host, as well as all intermediate hosts, if routing is to succeed in the cases of virtual circuit connections or bidirectional datagram transfers.

AUTHOR

`routing` was developed by the University of California, Berkeley.

FILES

`/etc/hosts`
`/etc/networks`

SEE ALSO

netstat(1), ifconfig(1m), route(1m).

NAME

scsi - Small Computer System Interface device drivers

DESCRIPTION

The Small Computer System Interface (SCSI) is an American National Standard for interconnecting computers and peripheral devices (see ANSI Std X3.131-199X, "SCSI-2"). The SCSI standard includes specifications for a variety of device types. This section describes the general SCSI interface for all SCSI device drivers. Information about specific device types can be found in the manual sections which describe SCSI peripheral device drivers for those device types.

The **SIOC_INQUIRY** ioctl is supported by all SCSI device drivers. This ioctl returns the SCSI device-specific INQUIRY command data. This data contains device identification and capability information. Since there have been multiple versions of the SCSI standard for inquiry data, multiple versions of the inquiry data declaration are provided. The SCSI-1 version is provided for backward compatibility only.

The **SIOC_CAPACITY** ioctl indicates the current device size. A device size is defined to be a logical block size and some number of logical blocks. The means of determining this device-size data is particular to the specific device type. Logical block size and/or number of logical blocks equal to zero indicates: the device size is unknown, the device is not currently capable of I/O operations, or I/O operations are not meaningful for the device.

The header file <**sys/scsi.h**> has useful information for SCSI devices. The following is included from <**sys/scsi.h**>:

```
#define     SIOC_INQUIRY            _IOR('S', 2, union inquiry_data)
#define     SIOC_CAPACITY           _IOR('S', 3, struct capacity)

/* SCSI-1 inquiry structure */
struct inquiry {
        unsigned char    dev_type;
        unsigned char    rmb:1;
        unsigned char    dtq:7;
        unsigned char    iso:2;
        unsigned char    ecma:3;
        unsigned char    ansi:3;
        unsigned char    resv:4;
        unsigned char    rdf:4;
        unsigned char    added_len;
        unsigned char    dev_class[3];
        char             vendor_id[8];
        char             product_id[16];
        char             rev_num[4];
        unsigned char    vendor_spec[20];
        unsigned char    resv4[40];
        unsigned char    vendor_parm_bytes[32];
};

/* SCSI-2 inquiry structure */
struct inquiry_2 {
        unsigned char    periph_qualifier:3;
        unsigned char    dev_type:5;
        unsigned char    rmb:1;
        unsigned char    dtq:7;
        unsigned char    iso:2;
        unsigned char    ecma:3;
        unsigned char    ansi:3;
        unsigned char    aenc:1;
        unsigned char    trmiop:1;
        unsigned char    resv1:2;
        unsigned char    rdf:4;
        unsigned char    added_len;
        unsigned char    resv2[2];
        unsigned char    reladr:1;
```

```
        unsigned char    wbus32:1;
        unsigned char    wbus16:1;
        unsigned char    sync:1;
        unsigned char    linked:1;
        unsigned char    resv3:1;
        unsigned char    cmdque:1;
        unsigned char    sftre:1;
        char             vendor_id[8];
        char             product_id[16];
        char             rev_num[4];
        unsigned char    vendor_spec[20];
        unsigned char    resv4[40];
        unsigned char    vendor_parm_bytes[32];
};

/* union for SIOC_INQUIRY ioctl */
union inquiry_data {
        struct inquiry   inq1;    /* SCSI-1 inquiry */
        struct inquiry_2 inq2;    /* SCSI-2 inquiry */
};

/* structure for SIOC_CAPACITY ioctl */
struct capacity {
        int   lba;
        int   blksz;
};
```

DEPENDENCIES

Series 300/400/700

The **SIOC_XSENSE** ioctl returns detailed information about device status and errors when such information is available. Since there have been multiple versions of the SCSI standard for sense (status) data, multiple versions of the sense data declaration are provided. The SCSI-1 and non-aligned versions are provided for backward compatibility only. If no new CHECK-CONDITION-caused REQUEST SENSE command data has been obtained since the last **SIOC_XSENSE** ioctl call, the **xsense_aligned.error_class** and **sense_2_aligned.error_code** fields will contain the value zero. Applications which require more accurate REQUEST SENSE data handling should use the SCSI device-control driver (see *scsi_ctl*(7)).

The following information is included from <**sys/scsi.h**>:

```
#define SIOC_XSENSE          _IOR('S', 7, union sense_data)

/* structure for SIOC_XSENSE ioctl */
union sense_data {
    struct xsense_aligned r_sense1a; /* SCSI and CCS devices */
    struct sense_2_aligned r_sense2a;    /* SCSI-2 devices */
};

/* structure for SCSI-1 and SCSI-CCS sense data */
struct xsense_aligned {
    unsigned char  valid:1;
    unsigned char  error_class:3;
    unsigned char  error_code:4;
    unsigned char  seg_num;
    unsigned char  parms:4;
    unsigned char  sense_key:4;
    unsigned char  lba[4];
    unsigned char  add_len;
    unsigned char  copysearch[4];
    unsigned char  sense_code;
    unsigned char  resv;
    unsigned char  fru;
    unsigned char  field;
    unsigned char  field_ptr[2];
```

```
    unsigned char  dev_error[4];
    unsigned char  misc_bytes[106];
};

/* structure for SCSI-2 sense data */
struct sense_2_aligned {
    unsigned char  info_valid:1;
    unsigned char  error_code:7;
    unsigned char  seg_num;
    unsigned char  filemark:1;
    unsigned char  eom:1;
    unsigned char  ili:1;
    unsigned char  resv:1;
    unsigned char  key:4;
    unsigned char  info[4];
    unsigned char  add_len;
    unsigned char  cmd_info[4];
    unsigned char  code;
    unsigned char  qualifier;
    unsigned char  fru;
    unsigned char  key_specific[3];
    unsigned char  add_sense_bytes[113];
};
```

Series 700

The **SIOC_EXCLUSIVE** ioctl may be used to obtain and release exclusive access. Exclusive access, which prevents simultaneous access by other applications, is required for some operations and may be desirable in other circumstances. The following exclusive access control arguments are supported:

0 Release exclusive access to logical unit (LUN).

1 Gain exclusive access to logical unit (LUN).

2 Release exclusive access to associated SCSI target.

3 Gain exclusive access to associated SCSI target.

4 Release exclusive access to associated SCSI bus.

5 Gain exclusive access to associated SCSI bus.

The **SIOC_MEDIUM_CHANGED** ioctl indicates when the media in a removable-media device may have changed. A value of "1" indicates the device media may have changed since the last **SIOC_MEDIUM_CHANGED** ioctl call. Note that only the first such call after a media change receives this indication. This means that media changes are likely to be missed if multiple applications are attempting to detect media changes. Exclusive access, obtained through use of the **SIOC_EXCLUSIVE** ioctl, can be used to avoid this problem.

The following information is included from **<sys/scsi.h>**:

```
#define SIOC_MEDIUM_CHANGED  _IOR('S', 42, int)
#define SIOC_EXCLUSIVE       _IOR('S', 68, int)
```

Series 800

The **SIOC_VPD_INQUIRY** ioctl allows access to detailed device specific information. The **page_code** field specifies which SCSI vital product data page is requested. The **page_buf** field is filled with the requested page data.

The following information is included from **<sys/scsi.h>**:

```
#define SIOC_VPD_INQUIRY  _IOWR('S', 10, struct vpd_inquiry)

/* union for SIOC_VPD_INQUIRY ioctl */
struct vpd_inquiry {
       char    page_code;       /* VPD page code              */
       char    page_buf[126];   /* buffer for VPD page info   */
};
```

WARNINGS

Use of devices that are not officially supported can cause data loss, system panics and device damage. HP-UX device drivers expect devices to be SCSI-2 compliant. Unsupported devices that are only SCSI-CCS compliant may work but their use is discouraged. Use of unsupported devices that are only SCSI-1 compliant is strongly discouraged.

Changing SCSI bus connectivity (recabling) while the system is running is not supported. Switching SCSI device power on or off while the device is connected to a system that does not support powerfail recovery is not supported. These activities are known to cause data loss and system panics.

On systems that support the **scsi_ctl** interface, the **SIOC_CMD_MODE**, **SIOC_SET_CMD**, and **SIOC_RETURN_STATUS** ioctls are obsolete (see *scsi_ctl*(7)). Direct manipulation of SCSI devices via the **scsi_ctl** interface provides a more functionally complete and easier-to-use means of low level SCSI device control (see *scsi_ctl*(7)).

Drivers that support only devices which have no meaningful size may not support the **SIOC_CAPACITY** ioctl. Total device size in bytes may exceed $2^{32}-1$ for some devices.

ERRORS

The following errors may result from a call to a SCSI device driver:

[EACCES] Required permission is denied for the the device or operation.

[ENXIO] If resulting from an open call, this indicates there is no device at the specified address. For other calls, this indicates the specified address is out of range or the device may no longer be accessed.

[EINVAL] If resulting from an open call, this indicates the device is not supported by the device driver (e.g. incorrect device type). For other calls, this indicates the request or some request argument is invalid.

[EBUSY] This indicates the device is not ready for use or that the requested operation conflicts with other operations (e.g. the device is currently open via another device driver or exclusive access is in effect).

[EIO] Indicates a SCSI protocol or communication problem has occurred, or that a SCSI command resulted in a non-good status.

Manual entries that describe specific SCSI peripheral device drivers may provide additional qualification of error results.

SEE ALSO

scsi_disk(7), scsi_tape(7), scsi_changer(7), scsi_ctl(7), diskinfo(1M).

NAME

scsi_changer - SCSI media changer device driver

DESCRIPTION

SCSI media changer devices mechanically move media between storage and usage locations. Each potential media location has a specific element address and is one of the following element types:

storage	A location to hold a unit of media not currently in use. Typically most media will be located in this type of element.
import/export	A location for inserting and removing media from the device. Movement of a unit of media to this type of location is in effect an eject operation. Movement of a unit of media from this type of location is a load operation.
data transfer	A location for accessing media data. This is generally the location of a device that reads and/or writes data on the media being handled by the media changer device. Movement to this type of location is a physical-media-mount operation. Movement from this type of location is a physical-media-unmount operation.
media transport	A location for media movement. Media is generally temporarily located in this type of element only during actual media movement.

The **SIOC_INIT_ELEM_STAT** ioctl causes the media changer device to take inventory. As a result, the media changer device determines the status of each and every element address, including the presence or absence of a unit of media.

The element addresses supported by a media changer device can be determined by use of the **SIOC_ELEMENT_ADDRESSES** ioctl. The first valid element address and the number of elements is indicated for each element type. These element addresses may be used as source and destination location arguments.

The **SIOC_ELEMENT_STATUS** ioctl indicates the status of an element. The element address for which status information is requested is specified via the **element** field. The resulting status data indicates the presence or absence of a unit of media in that element address as well as other information.

The **SIOC_RESERVE** and **SIOC_RELEASE** ioctls control access to element addresses. Depending on the device, reservations may limit operator control of those element addresses in the media changer device. Specific element addresses can be reserved to handle interlocking between multiple requesters if each requester has a unique reservation identification. The value zero in the **all_elements** field specifies that a single element address should be reserved or released. An element address reserved in this manner can not be reserved by another single element address reservation using a different reservation identification. The **reservation** field specifies the reservation identification. The **element** field specifies the element address to be reserved.

The value "1" in the **all_elements** field indicates that all element addresses should be reserved. The **reservation** and **element** fields should contain the value zero since these fields are not meaningful when reserving all element addresses. Reserving all element addresses is primarily useful for limiting operator control.

The **SIOC_MOVE_MEDIUM** and **SIOC_EXCHANGE_MEDIUM** ioctls reposition unit(s) of media. Depending of the source and destination element types, this may result in a media load, eject, mount, unmount, or simple repositioning. Media can be "flipped" using values of "1" in the **invert, invert_first,** or **invert_second** fields. The **SIOC_EXCHANGE_MEDIUM** ioctl repositions two different units of media. One unit of media is moved from the element specified by the **source** field to the element specified by the **first_destination** field. A second unit of media is moved from the element specified by the **first_destination** field to the element specified by the **second_destination** field. An exchange occurs if the **source** and **second_destination** fields are the same.

The following is included from **<sys/scsi.h>**:

```
#define SIOC_INIT_ELEM_STAT       _IO('S', 51)
#define SIOC_ELEMENT_ADDRESSES    _IOW('S', 52, struct element_addresses)
#define SIOC_ELEMENT_STATUS       _IOWR('S', 53, struct element_status)
#define SIOC_RESERVE              _IOW('S', 54, struct reservation_parms)
#define SIOC_RELEASE              _IOW('S', 55, struct reservation_parms)
#define SIOC_MOVE_MEDIUM          _IOW('S', 56, struct move_medium_parms)
```

```
#define SIOC_EXCHANGE_MEDIUM     _IOW('S', 57, struct exchange_medium_parms)

/* structure for SIOC_INIT_ELEM_STAT ioctl */
struct element_addresses {
        unsigned short  first_transport;
        unsigned short  num_transports;
        unsigned short  first_storage;
        unsigned short  num_storages;
        unsigned short  first_import_export;
        unsigned short  num_import_exports;
        unsigned short  first_data_transfer;
        unsigned short  num_data_transfers;
};

/* structure for SIOC_ELEMENT_STATUS ioctl */
struct element_status {
        unsigned short  element;          /* element address */
        unsigned char   full:1;           /* holds a a unit of media */
        unsigned char   reserved:1;       /* is currently reserved */
        unsigned char   except:1;         /* is in an abnormal state */
        unsigned char   access:1;         /* transport element accessible */
        unsigned char   export_enable:1;  /* allows media removal (eject) */
        unsigned char   import_enable:1;  /* allows media insertion (load) */
        unsigned char   resv1:2;
        unsigned char   resv2;
        unsigned char   sense_code;       /* info. about abnormal state */
        unsigned char   sense_qualifier;  /* info. about abnormal state */
        unsigned char   not_bus:1;        /* transfer device SCSI bus differs */
        unsigned char   resv3:1;
        unsigned char   id_valid:1;       /* bus_address is valid */
        unsigned char   lu_valid:1;       /* lun is valid */
        unsigned char   resv4:1;
        unsigned char   lun:3;            /* transfer device SCSI LUN */
        unsigned char   bus_address;      /* transfer device SCSI address */
        unsigned char   resv4;
        unsigned char   resv5:6;
        unsigned char   invert:1;         /* media in element was inverted */
        unsigned char   source_valid:1;   /* source_element is valid */
        unsigned short  source_element;   /* last storage location of medium */
        char            pri_vol_tag[36];  /* volume tag (device optional) */
        char            alt_vol_tag[36];  /* volume tag (device optional) */
        unsigned char   misc_bytes[168];  /* device specific */
};

/* structure for SIOC_RESERVE and SIOC_RELEASE ioctls */
struct reservation_parms {
        unsigned short  element;
        unsigned char   identification;
        unsigned char   all_elements;
};

/* structure for SIOC_MOVE_MEDIUM ioctl */
struct move_medium_parms {
        unsigned short  transport;
        unsigned short  source;
        unsigned short  destination;
        unsigned char   invert;
};

/* structure for SIOC_EXCHANGE_MEDIUM ioctl */
struct exchange_medium_parms {
        unsigned short  transport;
```

```
        unsigned short  source;
        unsigned short  first_destination;
        unsigned short  second_destination;
        unsigned char   invert_first;
        unsigned char   invert_second;
};
```

WARNING

Some media changer devices do not support the **SIOC_INIT_ELEM_STAT** and **SIOC_ELEMENT_STATUS** ioctls. For some of these devices, it may be possible to determine the full or empty status of an element address by attempting media movement using that element address as both the source and destination arguments of a media movement operation.

Many media changer devices do not support the **SIOC_EXCHANGE_MEDIUM** ioctl. For these devices, multiple **SIOC_MOVE_MEDIUM** ioctl operations may be used to accomplish the same results, provided a suitable temporary element address may be found.

SEE ALSO

scsi(7), autochanger(7), mknod(1M).

NAME

scsi_ctl - SCSI device control device driver

DESCRIPTION

SCSI devices should normally be controlled by a device-type-specific driver when the appropriate device-type-specific driver exists. Device-type-specific drivers, such as those for SCSI direct access (disk) and sequential access (tape) devices coordinate device and driver states to accomplish correct logical device behavior. The SCSI device-control driver enables use of SCSI devices and commands not normally supported by these device-type-specific drivers.

A successful scsi_ctl `open()` call requires no device I/O operations. This means that an `open()` call completes successfully even if there is no device at the indicated SCSI bus address. Once open, `ioctl()` calls can be used to change SCSI communication parameters or attempt SCSI commands and other SCSI operations. Since the SCSI device-control driver does not attempt to logically understand the target device, `read()` and `write()` calls are not supported.

Except where noted, the ioctls described here are available through all SCSI device drivers (including device-type-specific device drivers). Super-user privileges or device write permissions are required to use these ioctls through a SCSI device-type-specific device driver. All `reserved` fields in the data structures associated with these ioctls must be zero-filled.

SCSI Communication Parameters

SCSI communication parameters control features related to SCSI bus communication. Communication parameters are defined for three different scope levels: bus, target and logical unit number (LUN). Bus communication parameters apply to all targets connected to a specific bus. Target communication parameters apply to all LUNs associated with a specific target. LUN communication parameters apply to a specific LUN. SCSI communication parameters apply to all device drivers (both device-type-specific and device-control).

At power-up and after being reset, all SCSI devices and hosts communicate using asynchronous data transfers. Asynchronous data transfers use request (REQ) and acknowledge (ACK) signalling. The strict ordering of REQ and ACK signalling simplifies the communication protocol but limits I/O performance. A SCSI target and host pair may agree to use synchronous data transfers to increase I/O performance. Synchronous data transfers improve I/O performance by loosening the ordering requirements on REQs and ACKs. By allowing multiple outstanding REQs, signal propagation delays and temporary rate imbalances can be more efficiently tolerated. To make use of synchronous data transfers, a SCSI target and host must negotiate to determine mutually acceptable maximum-REQ-ACK-offset and maximum-data-transfer-rate parameters. The maximum-REQ-ACK-offset parameter indicates the maximum allowable number of outstanding REQs. The value zero is used to indicate asynchronous data transfer. Other values indicate synchronous data transfer. The appropriate value is generally dependent on the size of the receive data FIFO. High values tend to improve data transfer rates. The maximum-data-transfer-rate parameter indicates the "burst" data transfer rate (minimum allowable time between successive synchronous data transfers). A SCSI synchronous data transfer request (SDTR) message, which is used to initiate the negotiation process, is associated with the processing of a SCSI command.

At power-up and after being reset, all SCSI devices and hosts communicate using eight bit data transfers. A SCSI target and host pair may agree to use sixteen- or thirty-two-bit (wide) data transfers to increase I/O performance. To make use of wide data transfers, a SCSI target and host must negotiate to determine a mutually acceptable data transfer width parameter. A SCSI wide data transfer request (WDTR) message, used to initiate the negotiation process, is associated with the processing of a SCSI command.

Some SCSI devices are able to simultaneously manage multiple active commands. Such a device has a command queue which holds commands for processing. This command queuing may improve I/O performance by reducing the time spent by the device waiting for new commands from the host. Note that command queuing may not substantially improve I/O performance for devices that support "read-ahead" and "immediate-reporting" (see *scsi_disk*(2) and *scsi_tape*(2)). The SCSI device and host use command tags to correctly manage these multiple simultaneously active commands. At all times when command queuing is in effect, each active command being handled by a specific LUN has a unique command tag.

SCSI devices indicate their ability to support the special communication features described above in their SCSI INQUIRY command data. Normally the SCSI INQUIRY command data and negotiation protocols allow hosts and devices to determine the optimal communication parameters so that I/O performance is maximized. The current operating communication parameters may be determined by use of the:

SIOC_GET_LUN_PARMS, SIOC_GET_TGT_PARMS, and SIOC_GET_BUS_PARMS ioctls.

Occasionally, it may be desirable to limit SCSI communication parameters to work around a communication problem or to provide external insight in determining optimal parameters. SCSI communication parameter limit suggestions may be specified by use of the: SIOC_SET_LUN_LIMITS, SIOC_SET_TGT_LIMITS, and SIOC_SET_BUS_LIMITS ioctls. Note that there may be substantial differences between specified communication parameter limit suggestions and the corresponding actual current communication parameters being used for communication. These differences are a result of: device-type-specific driver capabilities, interface driver capabilities, interface hardware capabilities, device capabilities, delays due to the negotiation process, delays due to currently active commands, and delays due to commands waiting to be sent to devices. Note that communication parameter limit suggestions may not survive between **close()** and **open()** calls, when no SCSI device drivers (device-type-specific or device-control) have associated LUN(s) open.

The current SCSI communication parameter limit suggestions may be determined by use of the: SIOC_GET_LUN_LIMITS, SIOC_GET_TGT_LIMITS, and SIOC_GET_BUS_LIMITS ioctls.

Logical unit communication parameters may be managed by use of the: SIOC_GET_LUN_PARMS, SIOC_SET_LUN_LIMITS, and SIOC_GET_LUN_LIMITS ioctls. The SIOC_GET_LUN_PARMS ioctl indicates the current LUN communication parameter values. The **flags** field indicates what special communication features are in use. If the SCTL_ENABLE_TAGS flag is set, commands being sent to the corresponding LUN are permitted to be tagged in support of command queuing. If the SCTL_TAGS_ACTIVE flag is set, commands being sent to the corresponding LUN are currently being tagged for support of command queuing. The **max_q_depth** field indicates the current maximum number of simultaneously active commands the host might send to that LUN. Note that the SCTL_TAGS_ACTIVE flag may be set when the **max_q_depth** field contains the value "1" to indicate commands are being tagged but are still being serially processed. The SIOC_SET_LUN_LIMITS ioctl may be used to provide LUN communication parameter limit suggestions. The **flags** field indicates what special communication features should be used. Setting the SCTL_ENABLE_TAGS flag specifies that commands should be tagged. Clearing the SCTL_ENABLE_TAGS flag specifies that commands should not be tagged. The **max_q_depth** field specifies the maximum number of simultaneously active commands that should be attempted by the host. The SIOC_GET_LUN_LIMITS ioctl indicates the current LUN communication parameter limit suggestions.

Target communication parameters may be managed by use of the: SIOC_GET_TGT_PARMS, SIOC_SET_TGT_LIMITS, and SIOC_GET_TGT_LIMITS ioctls to any associated LUN. The SIOC_GET_TGT_PARMS ioctl indicates the current target communication parameter values. The **flags** field indicates what special communication features are in use. If the SCTL_ENABLE_SDTR flag is set, synchronous data transfer request negotiation with the corresponding target is permitted. If the SCTL_SDTR_DONE flag is set, the synchronous data transfer request negotiation process has been completed and the negotiation results are available in the **reqack_offset** and **xfer_rate** fields. If the SCTL_ENABLE_WDTR flag is set, wide data transfer request negotiation with the corresponding target is permitted. If the SCTL_WDTR_DONE flag is set, the wide data transfer request negotiation process has been completed and the negotiation results are available in the **width** field. The **width** field indicates the current data transfer bus width in bits. The **reqack_offset** field indicates the current maximum number of outstanding REQs being attempted. The value zero indicates asynchronous data transfer signalling is currently being used. The **xfer_rate** field indicates the current maximum "burst" data transfer rate in bytes per second. The SIOC_SET_TGT_LIMITS ioctl specifies the target communication parameter limit suggestions. The **flags** field specifies what special communication features should be used. Setting the SCTL_ENABLE_SDTR flag specifies that synchronous data transfer request negotiation with the corresponding target should be attempted when appropriate. Clearing the SCTL_ENABLE_SDTR flag specifies that synchronous data transfer request negotiation should not be attempted. Setting the SCTL_ENABLE_WDTR flag specifies that wide data transfer request negotiation with the corresponding target should be attempted when appropriate. Clearing the SCTL_ENABLE_WDTR flag specifies that wide data transfer request negotiation should not be attempted. The **max_width** field specifies maximum bus width that should be used for data transfers. The **max_reqack_offset** field specifies the maximum number of outstanding REQs that should be attempted during data transfers. The **max_xfer_rate** field specifies the maximum "burst" data rate that should be allowed during data transfers. The SIOC_GET_TGT_LIMITS ioctl indicates the current target communication parameter limit suggestions.

Bus communication parameters may be managed by use of the: SIOC_GET_BUS_PARMS, SIOC_SET_BUS_LIMITS, and SIOC_GET_BUS_LIMITS ioctls to any associated LUN. The

SIOC_GET_BUS_PARMS ioctl indicates the current bus communication parameter values. The **max_width** field indicates the maximum data transfer width that will be attempted for data transfers to any target device connected to the associated bus. The **max_reqack_offset** field indicates the maximum number of outstanding SM REQs that will be attempted during data transfers to any target device connected to the associated bus. The **max_xfer_rate** field indicates the maximum "burst" data transfer rate that will be attempted for data transfers to any target device connected to the associated bus. The SIOC_SET_BUS_LIMITS ioctl specifies the bus communication parameter limit suggestions for targets connected to the associated bus. The **max_width** field specifies the suggested maximum data transfer width that should be attempted for data transfers to any target device connected to the associated bus. The **max_reqack_offset** field specifies the maximum number of outstanding REQs that should be attempted during data transfers to any target device connected to the associated bus. The **max_xfer_rate** field specifies the maximum "burst" data transfer rate that should be attempted for data transfers to any target device connected to the associated bus. The SIOC_GET_BUS_LIMITS ioctl indicates the current bus communication parameter limit suggestions.

The following is included from <sys/scsi.h>:

```
/* SCSI communication parameter ioctls */
#define SIOC_GET_LUN_PARMS      _IOR('S', 58, struct sioc_lun_parms)
#define SIOC_GET_TGT_PARMS      _IOR('S', 59, struct sioc_tgt_parms)
#define SIOC_GET_BUS_PARMS      _IOR('S', 60, struct sioc_bus_parms)
#define SIOC_GET_LUN_LIMITS     _IOR('S', 61, struct sioc_lun_limits)
#define SIOC_GET_TGT_LIMITS     _IOR('S', 62, struct sioc_tgt_limits)
#define SIOC_GET_BUS_LIMITS     _IOR('S', 63, struct sioc_bus_limits)
#define SIOC_SET_LUN_LIMITS     _IOW('S', 64, struct sioc_lun_limits)
#define SIOC_SET_TGT_LIMITS     _IOW('S', 65, struct sioc_tgt_limits)
#define SIOC_SET_BUS_LIMITS     _IOW('S', 66, struct sioc_bus_limits)

struct sioc_lun_parms {
      unsigned int flags;
      unsigned int max_q_depth;     /* maximum active I/O's */
      unsigned int reserved[4];     /* reserved for future use */
};

struct sioc_lun_limits {
      unsigned int flags;
      unsigned int max_q_depth;
      unsigned int reserved[4];     /* reserved for future use */
};

struct sioc_tgt_parms {
      unsigned int flags;
      unsigned int width;           /* bits */
      unsigned int reqack_offset;
      unsigned int xfer_rate;       /* bytes/sec */
      unsigned int reserved[4];     /* reserved for future use */
};

struct sioc_tgt_limits {
      unsigned int flags;
      unsigned int max_reqack_offset;
      unsigned int max_xfer_rate;   /* bytes/sec */
      unsigned int max_width;       /* bits */
      unsigned int reserved[4];     /* reserved for future use */
};

struct sioc_bus_parms {
      unsigned int flags;           /* reserved for future use */
      unsigned int max_width;
      unsigned int max_reqack_offset;
      unsigned int max_xfer_rate;   /* bytes/sec */
      unsigned int reserved[4];     /* reserved for future use */
};
```

```
struct sioc_bus_limits {
      unsigned int flags;           /* reserved for future use */
      unsigned int max_width;
      unsigned int max_reqack_offset;
      unsigned int max_xfer_rate;   /* bytes/sec */
      unsigned int reserved[4];     /* reserved for future use */
};
```

SCSI Commands and Operations

The `SIOC_IO` ioctl allows an arbitrary SCSI command to be sent to a device. All details of the SCSI command protocol are handled automatically.

The following flags can be used to specify the `flags` field value:

`SCTL_READ`	Data-in phase expected if the `data_length` field is non-zero. The absence of this flag implies that a data-out phase is expected if the `data_length` field is non-zero.
`SCTL_INIT_SDTR`	synchronous data transfer request negotiations should be attempted with this command.
`SCTL_INIT_WDTR`	wide data transfer request negotiations should be attempted with this command.
`SCTL_NO_ATN`	device should be selected without attention (ATN). This implies that no SCSI message phase should be attempted with this command.

The `cdb` field specifies the SCSI command bytes. The number of command bytes is specified by the `cdb_length` field. These command bytes are sent to the target device during the SCSI command phase.

The address of the data area for the data phase of the SCSI command is specified by the `data` field. The `data_length` field specifies the maximum number of data bytes to be transfered. A zero-valued `data_length` indicates that no data phase should occur. Most SCSI commands with a data phase expect the data length information to be included somewhere in the command bytes. The caller is responsible for correctly specifying both the `data_length` field and any cdb data length values. The length may not be larger than `MAXPHYS`.

The `max_msecs` field specifies the maximum time, in milliseconds, that the device should need to complete the command. If this period of time expires without command completion, the system attempts recovery procedures to regain the device's attention. These recovery procedures may include device and bus reset operations. A zero value in the `max_msec` field indicates that the timeout period is infinite and the system should wait indefinitely for command completion. Note that very large (or infinite) timeout values can cause the SCSI bus (potentially the entire system) to "hang".

When the `SIOC_IO` ioctl call returns, all command processing has been completed. Most `SIOC_IO` ioctl calls will return zero (success). The resulting detailed ioctl data should be used to evaluate "success" or "failure" from the caller's perspective. The `cdb_status` field indicates the results of the cdb command. If the `cdb_status` field indicates a `S_CHECK_CONDITION` status, the `sense_status` field indicates the results of the SCSI `REQUEST SENSE` command used to collect the associated sense data. These status fields will contain one of the following values:

`SCTL_INVALID_REQUEST`	The SCSI command request is invalid and was not attempted.
`SCTL_SELECT_TIMEOUT`	The target device did not answer to selection by the host SCSI interface (the device does not exist or did not respond).
`SCTL_INCOMPLETE`	The device answered selection but the command was not completed (the device took too long or a communication failure occurred).
`S_GOOD`	Device successfully completed the command.
`S_CHECK_CONDITION`	Device indicated sense data was available.
`S_CONDITION_MET`	Device successfully completed the command and the requested (search or pre-fetch) operation was satisfied.
`S_BUSY`	Device indicated it was unable to accept the command because it is busy doing other operations.

S_INTERMEDIATE	Device successfully completed this command, which was one in a series of linked commands (not supported, see WARNINGS).
S_I_CONDITION_MET	Device indicated both S_INTERMEDIATE and S_CONDITION_MET (not supported, see WARNINGS).
S_RESV_CONFLICT	Device indicated the command conflicted with an existing reservation.
S_COMMAND_TERMINATED	Device indicated the command was terminated early by the host system.
S_QUEUE_FULL	Device indicated it was unable to accept the command because its command queue is currently full.

The **data_xfer** field indicates the number of data bytes actually transfered during the data phase of the cdb command. This field is valid only when the **cdb_status** field contains one of the following values: S_GOOD or S_CHECK_CONDITION. The **sense_xfer** field indicates the number of valid sense data bytes. This field is valid only when the **cdb_status** field contains the value S_CHECK_CONDITION and the **sense_status** field contains the value S_GOOD.

The **SIOC_ABORT** ioctl causes a SCSI **ABORT** message to be sent to the associated target. A SCSI **ABORT** message causes the associated target to terminate all active commands.

The **SIOC_RESET_DEV** ioctl causes a SCSI **BUS DEVICE RESET** message to be sent to the associated target. A SCSI **BUS DEVICE RESET** message causes the associated target to be reset (including clearing all active commands).

The **SIOC_RESET_BUS** ioctl causes the system to generate a SCSI bus reset condition on the associated bus. A SCSI bus reset condition causes all devices on the bus to be reset (including clearing all active commands on all devices).

Often it is necessary or useful to prohibit other SCSI commands while performing device-control operations. Normally this should be done by gaining exclusive access via the **SIOC_EXCLUSIVE** ioctl. Occasionally this is not possible (e.g. diagnostic operations on a device containing a mounted file system). Priority mode causes all device-type-specific driver I/O operations (e.g. file system I/O and virtual memory page swapping) and all SCSI device driver open calls (including device-control driver open calls) to the associated LUN to block. These I/O operations and open calls are blocked for the entire duration that priority mode is in effect. While priority mode is in effect only **SIOC_IO** operations should be attempted (these operations will not be blocked). The **SIOC_PRIORITY** ioctl controls the LUN priority mode. This ioctl is only available via the device-control driver. The value "1" enables priority mode. The value zero disables priority mode.

The header file <**sys/scsi.h**> has useful information for SCSI device control. The following is included from <**sys/scsi.h**>:

```
/* SCSI device control ioctls */
#define SIOC_IO                 _IOWR('S', 22, struct sctl_io)
#define SIOC_ABORT              _IO('S', 44)
#define SIOC_RESET_DEV          _IO('S', 16)
#define SIOC_RESET_BUS          _IO('S', 9)
#define SIOC_PRIORITY_MODE      _IOW('S', 67, int)

/* Structure for SIOC_IO ioctl */
struct sctl_io
{
        unsigned        flags;
        unsigned char   cdb_length;
        unsigned char   cdb[16];
        void            *data;
        unsigned        data_length;
        unsigned        max_msecs;
        unsigned        data_xfer;
        unsigned        cdb_status;
        unsigned char   sense[256];
        unsigned        sense_status;
        unsigned char   sense_xfer;
      unsigned char   reserved[64];
```

```
};
```

EXAMPLES

Assume that *fildes* is a valid file descriptor for a SCSI device. The first example attempts a SCSI **INQUIRY** command:

```
#include <sys/scsi.h>

struct sctl_io sctl_io;
#define MAX_LEN 255
unsigned char inquiry_data[MAX_LEN];

memset(sctl_io, 0, sizeof(sctl_io)); /* clear reserved fields */
sctl_io.flags = SCTL_READ;           /* input data is expected */
sctl_io.cdb[0] = 0x12;               /* could use CMDinquiry from scsi.h */
sctl_io.cdb[1] = 0x00;
sctl_io.cdb[2] = 0x00;
sctl_io.cdb[3] = 0x00;
sctl_io.cdb[4] = MAX_LEN;            /* allocation length in command */
sctl_io.cdb[5] = 0x00;
sctl_io.cdb_length = 6;              /* 6 byte command */
sctl_io.data = &inquiry_data[0];     /* data buffer location */
sctl_io.data_length = MAX_LEN;       /* maximum transfer length */
sctl_io.max_msecs = 10000;           /* allow 10 seconds for command */
if (ioctl(fildes, SIOC_IO, &sctl_io) < 0)
{
        /* request was invalid */
}
```

The following example attempts a SCSI **TEST UNIT READY** command and checks to see if the device is ready, not ready, or in some other state.

```
#include <sys/scsi.h>

struct sctl_io sctl_io;

memset(sctl_io, 0, sizeof(sctl_io)); /* clear reserved fields */
sctl_io.flags = 0;                   /* no data transfer is expected */
sctl_io.cdb[0] = 0x00;               /* could use CMDtest_unit_ready */
sctl_io.cdb[1] = 0x00;
sctl_io.cdb[2] = 0x00;
sctl_io.cdb[3] = 0x00;
sctl_io.cdb[4] = 0x00;
sctl_io.cdb[5] = 0x00;
sctl_io.cdb_length = 6;              /* 6 byte command */
sctl_io.data = NULL;                 /* no data buffer is provided */
sctl_io.data_length = 0;             /* no data should be transfered */
sctl_io.max_msecs = 10000;           /* allow 10 seconds for command */
if (ioctl(fildes, SIOC_IO, &sctl_io) < 0)
{
        /* request was invalid */
}
else if (sctl_io.cdb_status == S_GOOD)
{
        /* device is ready */
}
else if (sctl_io.cdb_status == S_BUSY ||
       (sctl_io.cdb_status == S_CHECK_CONDITION &&
        sctl_io.sense_status == S_GOOD &&
          sctl_io.sense_xfer > 2 &&
          (sctl_io.sense[2] & 0x0F) == 2))  /* could use sense_data */
{
        /* device is not ready */
```

```
}
else
{
        /* unknown state */
}
```

WARNINGS

Incorrect use of SCSI device-control operations (even those attempting access to non-existent devices) can cause data loss, system panics, and device damage.

The **SIOC_EXCLUSIVE** ioctl should be used to gain exclusive access to a device prior to attempting **SIOC_IO** commands. If exclusive access is not obtained, **SIOC_IO** commands will be intermixed with device-type-specific driver commands, which can lead to undesirable results.

Device-type-specific drivers have the to ability to veto **SIOC_IO** commands that would be inappropriate or troublesome. However, since not all such operations are known and detected, care should be exercised to avoid disrupting device-type-specific drivers when using commands that modify internal device states.

It is very easy to cause system deadlock through incorrect use of the **SIOC_PRIORITY_MODE** ioctl. Normally it is necessary to lock the calling process into memory (see *plock*(2)) prior to enabling priority mode.

Most SCSI commands have a logical unit number (LUN) field. SCSI implementations on the HP-UX operating system select logical units via the SCSI **IDENTIFY** message. The LUN portion of the cdb should normally be set to zero, even when the LUN being accessed is not zero.

Use of linked commands is not supported.

Most SCSI commands with a data phase expect the data length information to be included somewhere in the command bytes. Both the **data_length** field and any cdb data length values must be correctly specified to get correct command results.

Very large (or infinite) timeout values can cause the SCSI bus (potentially the entire system) to "hang".

Device and/or bus reset operations can be used to regain a device's attention when a timeout expires.

Resetting a device can cause I/O errors and/or loss of cached data. This can result in loss of data and/or system panics.

Obtaining SCSI INQUIRY data by use of the **SIOC_INQUIRY** ioctl instead of by use of the **SIOC_IO** ioctl is generally preferable since SCSI implementations on the HP-UX operating system synchronize access of inquiry data during driver open calls.

Since communication parameters may be impacted by device-type-specific driver capabilities, device-type-specific driver use may result in communication parameter changes.

The **SIOC_CAPACITY** ioctl is not supported by the SCSI device-control driver because the meaning of capacity is device-type-specific.

SEE ALSO

scsi(7), mknod(1M).

NAME

scsi_disk - SCSI direct access (disk) device driver

DESCRIPTION

This section describes the interface for access of SCSI disk, CD-ROM and optical disk devices through the character special device driver.

SCSI direct access devices store a sequence of data blocks. Each direct access device has a specific device size consisting of a number of data blocks and a logical block size. All data blocks have the same logical block size. Since I/O operations must have a size that is an integral number of blocks, one logical block size is the smallest possible I/O quantity. The device block size can be determined through use of the `DIOC_DESCRIBE` and `SIOC_CAPACITY` ioctls (see *disk*(7) and *scsi*(7)). A direct access device that is not ready for use, whether due to no media installed or another reason, is interpreted to mean the device has zero size. An `open()` call to such a device succeeds, but subsequent `read()` and `write()` calls fail.

To improve performance, many SCSI disk devices have caches. These caches can be used for both read and write operations. Read cache use, called "read ahead", causes the disk drive to read data in anticipation of read requests. Read ahead is only apparent to users in the increased performance that it produces. Write cache use is called "immediate reporting". Immediate reporting increases I/O performance by reporting a completed write status before the data being written is actually committed to media. If the subsequent physical write operation fails to successfully complete, data may be lost. Physical write failures due to media defects are largely eliminated by use of automatic sparing in disk drives. Power failure between immediate reporting and media commit can result in cached data being lost. However, the period of time between these events is typically relatively small, making such losses unlikely. The `SIOC_GET_IR` ioctl can be used to determine if immediate-reporting functionality is currently being used by the device. The value "1" indicates immediate reporting is enabled. The value zero indicates immediate reporting is disabled. The `SIOC_SET_IR` ioctl can be used to enable or disable immediate reporting. A zero value disables immediate reporting. The value "1" enables immediate reporting.

Most SCSI removable media disk devices support "prevent" and "allow" media-removal commands. To avoid data corruption and data accessibility problems, media removal is prevented for the entire duration a removable media disk device is open. Because media removal is not supported, the `SIOC_MEDIUM_CHANGED` ioctl is not supported.

The header file <`sys/scsi.h`> has useful information for direct access device control. The following is included from <`sys/scsi.h`>:

```
/* ioctl support for SCSI disk devices */
#define SIOC_GET_IR             _IOR('S', 14, int)
#define SIOC_SET_IR             _IOW('S', 15, int)
```

DEPENDENCIES

Series 300, 400, and 700

The `SIOC_FORMAT` ioctl reformats the entire media surface. Exclusive access to the device, obtained through use of the `DIOC_EXCLUSIVE` ioctl (see *disk*(7)), is required prior to reformatting to ensure that other applications are not affected. The `fmt_optn` field can be used to select the desired media geometry. Only one media geometry is supported on most devices. The value zero should be used for these devices. The value zero can also be used to select the default geometry on devices that support multiple media geometries. The interleave field can be used to specify sector interleaving. The value zero specifies that an appropriate default interleave should be used.

The following series-specific information is included from <`sys/scsi.h`>:

```
#define SIOC_FORMAT         _IOW('S', 6, struct sioc_format)
struct sioc_format {
        short     fmt_optn;
        short     interleave;
};
```

Series 800

The `SIOC_FORMAT` ioctl reformats the entire media surface. Exclusive access to the device, obtained through use of the `DIOC_EXCLUSIVE` ioctl (see *disk*(7)), is required prior to reformatting to ensure that other applications are not affected. The integer value of the `SIOC_FORMAT` ioctl is used to specify a sector interleave. The value zero specifies that an appropriate default interleave should be used.

The following series specific information is included from <sys/scsi.h>:

```
#define SIOC_FORMAT        _IOW('S', 6, int)
```

Optical Disk Devices

The SIOC_VERIFY_WRITES ioctl controls the write mode. Normally written data is assumed to be correctly stored on the media. Verify-writes mode causes verification of written data to ensure that data has been correctly written. Verification can substantially reduce write performance and is not generally needed. The SIOC_VERIFY_WRITES ioctl can be used to enable or disable write verification. A zero value disables write verification. The value "1" enables write verification. Although write verification is primarily intended for optical media, some systems may support write verification on normal disk devices.

The SIOC_VERIFY ioctl verifies that a media area contains valid data (has been correctly written). A media area verified in this manner should not cause I/O errors when reading is attempted. The media area to be verified is specified via the start_lba and block_cnt fields. Although verification is primarily intended for optical media, some systems may support verify operations on normal disk devices.

The SIOC_WRITE_WOE ioctl controls the write mode used for magneto-optical disk devices. Normally magneto-optical write operations require two physical head passes. The first pass erases the media area to be written. The second pass actually writes the data. Write-without-erase mode dramatically increases write performance by skipping the first (erase media area) pass. To ensure that the correct data results, it is essential that write-without-erase operations be performed only on media that is known to be blank (previously erased or never used). The SIOC_WRITE_WOE ioctl can be used to enable or disable write-without-erase. A zero value disables write-without-erase. The value "1" enables write-without-erase.

The SIOC_ERASE ioctl allows media areas to be explicitly erased. The media area to be erased is specified via the start_lba and block_cnt fields. Media areas erased in this manner can be written using write-without-erase mode. Note that an erased media area is different from a media area written with some data values (e.g. zeros). An erased media area should not be read. Attempting to read an erased media area generally results in an I/O error.

The SIOC_VERIFY_BLANK ioctl verifies that a media area has been erased and is suitable for being written using write-without-erase mode. The media area to be verified is specified via the start_lba and block_cnt fields.

The following optical disk device specific information is included from <sys/scsi.h>:

```
#define SIOC_WRITE_WOE                  _IOW('S', 17, int)
#define SIOC_VERIFY_WRITES              _IOW('S', 18, int)
#define SIOC_ERASE                      _IOW('S', 19, struct scsi_erase)
#define SIOC_VERIFY_BLANK               _IOW('S', 20, struct scsi_verify)
#define SIOC_VERIFY                     _IOW('S', 21, struct scsi_verify)

/* structure for SIOC_ERASE ioctl */
struct scsi_erase {
        unsigned int    start_lba;
        unsigned short  block_cnt;
};

/* structure for SIOC_VERIFY_BLANK and SIOC_VERIFY ioctls */
struct scsi_verify {
        unsigned int    start_lba;
        unsigned short  block_cnt;
};
```

WARNINGS

Although disk devices have historically had small (typically 512 byte) block sizes, some disk devices (such as optical disks and disk arrays) have relatively large block sizes. Applications using direct raw disk access should use the DIOC_DESCRIBE or SIOC_CAPACITY ioctls to determine the appropriate minimum I/O size.

Media removal and insertion while a disk device is open is unsupported and unpredictable. Circumventing prevention of media removal should not be attempted. Device capacity changes resulting from subsequent media changes may not be recognized.

Often larger I/O operation sizes are expected to be more efficient. However, SCSI disk I/O operations that are large relative to the device's cache can result in insufficient cache space for the device to maintain full-media-speed data transfer rates. This can result in decreased I/O performance relative to smaller I/O sizes.

SEE ALSO

scsi(7), disk(7), mediainit(1M), mknod(1M).

NAME

scsi_tape - SCSI sequential access (tape) device driver

DESCRIPTION

SCSI sequential-access (tape) devices store a sequence of data blocks. Data can be read and written using either fixed or variable sized block mode. If supported by the device, variable sized block mode is normally used (even when all blocks are the same size). Fixed sized block mode is generally only used for tape devices which do not support variable sized blocks. Fixed sized block mode can be used on some tape devices which support variable sized blocks to increase I/O performance.

Generally SCSI tape devices are controlled through the *mt*(7) generic tape device interface. This section describes features that are specific to SCSI tape devices.

The **SIOC_CAPACITY** ioctl (see *scsi*(7)) can be used to determine remaining tape capacity for some tape devices. The **blksz** field indicates the "natural" block size of the device. This value may or may not be the current block size of the device. The number of blocks, indicated by the **lba** field, is an estimate of how much data can be written on the remaining media. A zero size is returned for devices that do not provide remaining-capacity information. The quantity of data that can actually be written may be higher or lower than indicated, depending on such factors as block size, media defects, data compression, and ability to maintain streaming.

To improve performance, most SCSI tape devices have caches. Read-cache use, called "read ahead", causes the tape drive to read data in anticipation of read requests. Read ahead is only apparent to users in the increased performance that it produces. Write-cache use is called "immediate reporting". Immediate reporting increases I/O performance by reporting a completed write status before the data being written is actually committed to media. This allows the application program to supply additional data so that continuous media motion, called "streaming", can be achieved. The **SIOC_GET_IR** ioctl can be used to determine if immediate-reporting functionality is currently being used by the device. The value "1" indicates immediate reporting is enabled. By default, the device driver attempts to enable immediate reporting. The **SIOC_SET_IR** ioctl can be used to explicitly enable or disable immediate reporting. A zero value disables immediate reporting. The value "1" enables immediate reporting. The **MTIOCTOP** ioctl **MTNOP** command can be used to cause any cached data to be written (committed) to media. Note that the device immediate reporting mode set by the **SIOC_SET_IR** ioctl survives between **close()** and **open()** calls, but not through system reboot.

The **SIOC_GET_BLOCK_SIZE** ioctl indicates the device's current block size. A block size of zero indicates the device is in variable-sized-block mode. A non-zero block size indicates the device is in fixed-sized-block mode.

The **SIOC_SET_BLOCK_SIZE** ioctl changes the current block size to the specified number of bytes. Setting the block size to zero specifies that variable-sized-block mode should be used. Any non-zero block size specifies that fixed-sized-block mode should be used. By default, the device driver attempts to set the block size to zero during open. If variable-sized-block mode is not supported by the device, the driver selects an appropriate block size for fixed-sized-block mode use. Note that the device block size set by the **SIOC_SET_BLOCK_SIZE** ioctl survives between **close()** and **open()** calls, but not through system reboot.

The **SIOC_GET_BLOCK_LIMITS** ioctl indicates the device's maximum and minimum fixed block-size limits. The device's minimum fixed block size is indicated by the **min_blk_size** field. The **max_blk_size** field contains the smaller of the maximum block size supported by the device and the maximum block size supported by the system (**MAXPHYS**). This is the largest valid block size for the specific combination of device, driver, and host system being used.

The **SIOC_GET_POSITION** ioctl can be used to determine the current media position for some devices. For devices that support this capability, the resultant value can be used to reposition the media to the same position in the future.

The **SIOC_SET_POSITION** ioctl can be used to cause media repositioning on some devices. For devices that support this capability, media repositioning via this mechanism can generally be completed more quickly than might be similarly accomplished using record, filemark, or setmark spacing. The argument value specified should be the result of a previous **SIOC_GET_POSITION** for that media volume.

The following is included from **<sys/scsi.h>**:

```
/* ioctl support for SCSI tape commands */
#define SIOC_GET_IR              _IOR('S', 14, int)
#define SIOC_SET_IR              _IOW('S', 15, int)
#define SIOC_GET_BLOCK_SIZE      _IOR('S', 30, int)
#define SIOC_SET_BLOCK_SIZE      _IOW('S', 31, int)
#define SIOC_GET_BLOCK_LIMITS    _IOW('S', 32, struct scsi_block_limits)
#define SIOC_GET_POSITION        _IOR('S', 33, int)
#define SIOC_SET_POSITION        _IOW('S', 34, int)

/* structure for SIOC_GET_BLOCK_LIMITS ioctl */
struct scsi_block_limits {
        unsigned min_blk_size;
        unsigned max_blk_size;
};
```

WARNINGS

SCSI bus and device resets cause some devices to reposition media to beginning-of-tape (BOT). This unintentional media repositioning can cause loss of data. The scsi_tape driver causes the first subsequent **open**( ) attempt to fail as an indication of potential data loss.

The **scsi_tape** driver does not write filemarks at close if the media has been programatically repositioned. Applications that reposition the media prior to closing the device should write any required tapemarks.

SEE ALSO

scsi(7), mt(7), mknod(1M).

NAME

socket - Interprocess communications

DESCRIPTION

Sockets are communication endpoints that allow processes to communicate either locally or remotely. They are accessed by means of a set of system calls (see *socket*(2)).

The following `ioctl()` requests are defined in <`sys/ioctl.h`> (see *ioctl*(2)):

`FIOSNBIO` If the int with the address *arg* is non-zero, the socket is put into non-blocking mode. Otherwise, the socket is put into blocking mode. Blocking mode is the default. The `FIONBIO` request is equivalent to the `FIOSNBIO` request, although using `FIONBIO` is not recommended. See *accept*(2), *connect*(2), *recv*(2), and *send*(2) for an explanation of how non-blocking mode is used.

`FIONREAD` For SOCK_STREAM sockets, the number of bytes currently readable from this socket is returned in the integer with the address *arg*. For SOCK_DGRAM sockets, the number of bytes currently readable, plus the size of the *sockaddr* structure (defined in <`sys/socket.h`>), is returned in the integer with the address *arg*.

`SIOCATMARK` For SOCK_STREAM TCP sockets, on return the integer with the address *arg* is non-zero if the inbound TCP stream has been read up to where the out-of-band data byte starts; otherwise the inbound TCP stream has not yet been read up to where the out-of-band data byte starts. For sockets other than SOCK_STREAM TCP sockets, on return the integer with the address *arg* is always zero.

`SIOCSPGRP` This request sets the process group or process ID associated with the socket to be the value of the integer with the address *arg*. A process group or process ID associated with the socket in this manner is signaled when the state of the socket changes: `SIGURG` is delivered upon the receipt of out-of-band data; `SIGIO` is delivered if the socket is asynchronous, as described in `FIOASYNC` below. If the value of the integer with the address *arg* is positive, the signal is sent to the process whose process ID matches the value specified. If the value is negative, the signal is sent to all the processes that have a process group equal to the absolute value of the value specified. If the value is zero, no signal is sent to any process. It is necesary to issue this request with a non-zero integer value to enable the signal delivery mechanism described above; the default for the process group or process ID value is zero.

`SIOCGPGRP` This request returns the process group or process ID associated with the socket in the integer with the address *arg*. See the explanation for `SIOCSPGRP` above for more details on the meaning of the integer value returned.

`FIOASYNC` If the integer whose address is *arg* is non-zero, this request sets the state of the socket as asynchronous. Otherwise, the socket is put into synchronous mode (the default). Asynchronous mode enables the delivery of the `SIGIO` signal when:

- New data arrives, or
- For connection-oriented protocols, whenever additional outgoing buffer space becomes available, or when the connection is established or broken.

The process group or process ID associated with the socket must be non-zero in order for `SIGIO` signals to be sent; the signal is delivered according to the semantics of `SIOCSPGRP` descibed above.

The *fcntl*(2) `O_NDELAY` and `O_NONBLOCK` flags (defined in <`sys/file.h`>) are supported by sockets. If the `O_NONBLOCK` flag is set, the socket is put into POSIX-style non-blocking mode. If the `O_NDELAY` flag is set, the socket is put into non-blocking mode. Otherwise, the socket is put into blocking mode. Blocking mode is the default. See *accept*(2), *connect*(2), *recv*(2), and *send*(2) for an explanation of how these form of non-blocking mode is used.

Since both the `fcntl()` `O_NONBLOCK` and `O_NDELAY` flags and `ioctl()` `FIOSNBIO` requests are supported, some clarification on how these features interact is necessary. If the `O_NONBLOCK` or `O_NDELAY` flag has been set, `recv()` and `send()` requests behave accordingly, regardless of any `FIOSNBIO` requests. If neither the `O_NONBLOCK` flag nor the `O_NDELAY` flag has been set, `FIOSNBIO` requests control the the behavior of `recv()` and `send()`.

DEPENDENCIES

This entry describes the use of the TCP protocol as it applies to the Berkeley Interprocess Communication utility. It does not apply to the use of TCP for the NetIPC utility. Refer to the *NetIPC Programmer's Guide* for information about NetIPC.

AF_CCITT Only

Only the `FIOSNBIO`, `FIONREAD`, `SIOCGPGRP`, and `SIOCSPGRP` `ioctl()` requests are defined for `af_ccitt` sockets. See *socketx25*(7).

AUTHOR

`socket` was developed by the University of California, Berkeley.

SEE ALSO

fcntl(2), getsockopt(2), ioctl(2), socket(2), socketx25(7).

(Requires Optional LAN/X.25 Software)

NAME

socketx25 - Interprocess communications via AF_CCITT sockets

DESCRIPTION

Sockets are communication endpoints that allow processes to communicate either locally or remotely. They are accessed by means of a set of system calls (see *socket*(2)). The discussion below describes *ioctl*(2) calls available when directly accessing the X.25 Packet level (level 3) via sockets.

That is, when the sockets are in the address family AF_CCITT. Refer to the *af_ccitt*(7F) manual entry for details.

COMMAND FUNCTIONS

The following *ioctl*(2) requests are defined in **<x25/x25ioctls.h>**:

X25_RD_HOSTADR

Loads the X.121 address of an X.25 interface on the local host into an **x25addrstr struct**. Example:

```
struct x25addrstr bind_addr;
bind_addr.ifname[0] = '\0';
error = ioctl(s,X25_RD_HOSTADR,&bind_addr);
```

X25_WR_MASK_DATA

Sets a bit mask to be **and**ed with Protocol-IDs for address-matching during a *bind*() call. Each byte of the bit mask is set in an entry in the **x25_mask[ ]** array in an **x25_mask_data** data structure. The length (in bytes) of the bit mask is stored in the **x25_masklen** field of an **x25_mask_data** data structure. **x25_mask_data** is defined in the **x25str.h** header file. For example, to set an address mask accepting Protocol-IDs in the range 0xc033 to 0xcf33:

```
int s;
struct x25_mask_data in_parms;
in_parms.x25masklen = 2;
in_parms.x25_mask[0] = 0xf0;
in_parms.x25_mask[1] = 0xff;
error = ioctl(s, X25_WR_MASK_DATA, &in_parms);
```

X25_RESET_VC

Resets a Virtual Circuit (VC) associated with a particular socket, which might cause data loss. Issuing this call sends an out-of-band event (OOB_VC_RESET) to the peer user. If blocking I/O is being used, *ioctl*( ) blocks until an acknowledgement is received from the remote node. Example:

```
error = ioctl(s, X25_RESET_VC, 0);
```

X25_SEND_TYPE

Sets the D-bit or the Q-bit in a sequence of X.25 packets. Also controls whether the next *send*(2) will send a complete or a partial X.25 message via the More-Data-To-Follow (MDTF) bit setting. On Series 700/800 systems, setting the D-bit lets the sending process indicate that it requires acknowledgement when the end of a message has been received. If a Series 700/800 system receives a data packet with the D-bit set and the socket has been set to permit D-bit operations, it will ackowledge only when the receiving process has read the data. If a Series 300/400 system receives a data packet with the D-bit set, it acknowledges automatically, without guaranteeing that the receiving process has read the data. Any attempt to set the D-bit on a Series 300/400 system is rejected and the error [EINVAL] is returned.

Setting the Q-bit indicates that the data being sent is significant to a device connected to the remote host. Set these bits by shifting the appropriate bit mask to the left one bit and ORing the mask with a user-defined *send_type* variable as in this example:

```
int s, send_type, error;
send_type |= 1<<X25_Q_BIT;/* set Q*/
```

or

```
send_type |= 1<<X25_D_BIT;/* set D*/
error = ioctl(s, X25_SEND_TYPE, &send_type);
```

To use the D-bit or Q-bit, they must be set before the first *send*() in a message. Use X25_SEND_TYPE to set the D-bit prior to connect, or X25_SEND_CALL_ACCEPT to set the connection to permit use of D-bit data sends.

To send a partial X.25 message, MDTF bit should be set to 1. To send the final segment or to flush out a complete message, the MDTF bit should be reset to 0 before the *send*(2). Example:

```
int     socket, send_type, first_msglen,
        second_msglen, third_msglen;
u_char*first_msg, second_msg, third_msg;

/* set MDTF indicator to indicate multiple message
 * fragments ....
 */
send_type = 1 << X25_MDTF_BIT;
ioctl(socket,X25_SEND_TYPE,&send_type);

/* send the first two message fragments ....
 */
return = send(socket, first_msg, first_msglen, 0);
return = send(socket, second_msg, second_msglen, 0);

/* now clear the MDTF indicator ....
 * and send the third (final)  message fragment ...
 */
send_type = 0;
ioctl(socket,X25_SEND_TYPE,&send_type);
return = send(socket, third_msg, third_msglen, 0);
```

NOTE: The values of the Q bit and the D bit must remain constant throughout a complete X.25 message sequence. Therefore, the values of the D and Q bits are latched from the first X25_SEND_TYPE *ioctl*(), and any attempt to change them during transfer is ignored.

X25_NEXT_MSG_STAT

Returns status information on the next-available (unread) message, such as its size, whether message is partial or complete, whether the D and Q bits are set, and whether Call-user or clear data is available. This information is returned in an **x25_msg_stat**struct. Example:

```
/* Globals
 */
int   MDTF;
char  *malloc();
......

int                 socket, count;
unsigned int        length;
char                *fragment;
struct x25_msg_stat next_msg_stat;

ioctl(socket, X25_NEXT_MSG_STAT, &next_msg_stat);
if (next_msg_stat.x25_msg_flags &
(1 << X25_MDTF_BIT)) {
/* this is a fragment of the complete X.25
message ...
 */
MDTF = TRUE;
printf( More Data To Follow ....0);"
} /* if next_msg_stat .... */
else {
/* MDTF indicator is not set, use the MDTF
 * global to determine if this the final
 * fragment in a message sequence or a complete
 * message sequence ....
 */
```

(Requires Optional LAN/X.25 Software)

```
if (MDTF == TRUE) {
/* this is the final fragment in the message
 * sequence ....
 */
printf( Final fragment of X.25 message
printf( .0);"
} /* if MDTF == TRUE .... */
else {
/* this is a complete X.25 message sequence.
 */
printf( Complete X.25 message received.0);"
} /* else MDTF == FALSE .... */

/* now set MDTF to FALSE ....
 */
MDTF = FALSE;
/* get state of D and Q bits
 */
if (next_msg_stat.x25_msg_flags & (1<<X25_D_BIT)) {
/* D bit for this sequence has been set ....
 */
printf( , D bit set for
}
if (next_msg_stat.x25_msg_flags & (1<<X25_Q_BIT)) {
/* Q bit for this sequence has been set ....
 */
printf( , Q bit set for
}

} /* else next_msg_stat .... */

/* now get enough memory to receive the message ....
 */
length = (unsigned int)next_msg_stat.x25_msg_size;
message = malloc(length);
if (message == NULL) {
/* malloc couldn't get memory ....
 */
}
count = recv(socket, message, (int) length, 0);
```

Note that the Q bit value remains the same during subsequent receives of the fragments in a complete X.25 message sequence. The D bit status for a complete X.25 message sequence are returned before the *recv*() of the FINAL fragment. The D bit status is also returned before the *recv*() of a complete X.25 message sequence.

X25_WR_CAUSE_DIAG

Sets the cause code and diagnostic code for the next user-initiated RESET or CLEAR packet. Cause and diagnostic codes are set in the **x25_cause_diag** data structure defined in the **x25str.h** header file. The cause code, if non-zero, will have its most significant byte set by the X.25 subsystem as it appears in the X.25 packet. Set the cause and diagnostic codes as follows:

```
struct x25_cause_diag diag;
int s; /* socket */

diag.x25_cd_cause = cause_code;
diag.x25_cd_diag = diagnostic_code;
error = ioctl(s, X25_WR_CAUSE_DIAG, &diag);
```

X25_CALL_ACPT_APPROVAL

Gives user the option to accept incoming calls on a *listen ()* socket. When the X25_CALL_ACPT_APPROVAL *ioctl*() call is issued, a new *accept*() socket is created whenever a valid call comes in, but the call is not accepted at the X.25 level until an X25_SEND_CALL_ACEPT

ioctl() call is issued. Until the X25_SEND_CALL_ACEPT *ioctl*() call is issued, no data can be sent or received on the new socket. If the application does not want to accept the call, the circuit can be cleared with *close*(). Call acceptance cannot be turned-off for a socket once the X25_CALL_ACPT_APPROVAL *ioctl*() is issued. The call can be issued as follows:

```
error = ioctl(s, X25_CALL_ACPT_APPROVAL, 0);
```

where **s** is a listen socket; that is, a *listen*() system call has been issued on it.

X25_SEND_CALL_ACEPT
Accepts a call on a new socket returned by *accept*() when the listen socket has an X25_CALL_ACPT_APPROVAL *ioctl*() issued against it; otherwise it is illegal. Once X25_SEND_CALL_ACEPT is issued against a new socket, the socket is in a data-transfer state. Accepting the new socket is described below:

```
error = ioctl(new_s, X25_SEND_CALL_ACEPT, 0);
```

where **new_s** is the new socket created by *accept*().

X25_SETUP_PVC
Binds a socket to a Permanent Virtual Circuit (PVC). The user must create a socket, then enter the X.25 interface name and the PVC's Logical Channel Number (LCI) in the **x25_setup_pvc_str** data structure defined in the **<x25/x25str.h>** header file, before issuing the X25_SETUP_PVC *ioctl*() call. This *ioctl*() call is the only call required for setting-up a connection on a PVC; *listen*(), *accept*(), and *connect*() are not required for establishing a connection on a PVC. Once the *ioctl*() call completes successfully, the socket is in a connected state and data can be transmitted over the connection. For example,

```
struct x25_setup_pvc_str pvc_str;
int pvc_so;
pvc_so = socket(AF_CCITT, SOCK_STREAM, 0);
/* Set the interface name in pvc_str.ifname.
 * Store the LCI in pvc_str.lci. */
error = ioctl(pvc_so, X25_SETUP_PVC, &pvc_str);
```

X25_RD_USER_DATA
Reads data from the Call-user data field of a CALL CONNECTED or INCOMING CALL packet, or reads the data field of a CLEAR packet. Use the X25_NEXT_MSG_STAT *ioctl*() call to determine that CALL or CLEAR data is available, then issue the X25_RD_USER_DATA *ioctl*() call. The call transfers user data to the **x25_userdata** data structure defined in **<x25str.h>**, up to 126 bytes at a time. If more than 126 bytes of user data is available, the application should loop, calling X25_NEXT_MSG_STAT and X25_RD_USER_DATA until all the user data is read. This call is useful for fast-select. The example below calls X25_NEXT_MSG_STAT to determine whether CALL-user data available, then reads at most 126 bytes of data.

```
struct x25_userdata userdata;
struct x25_msg_stat msgstat;
error = ioctl(s, X25_NEXT_MSG_STAT, &msgstat);
if (msgstat.x25_msg_flags & ( 1 << X25_CA_DATA_AVAIL))
error =  ioctl(s, X25_RD_USER_DATA, &userdata);
```

X25_WR_USER_DATA
Writes to the Call-user data field in the following situations:

- Before issuing a *connect*() call,
- Before issuing an *accept*() call when Call-accept approval is in effect,
- Before issuing a *close*() call (which sends a CLEAR packet over the connection); otherwise, it has no effect.

If more than 16 bytes of Call and Clear data are written, the fast-select facilities code must be set with the X25_WR_FACILITIES *ioctl*() call. Only 126 bytes of Call and Clear data can be written per *ioctl*() call. If the total amount of user data is greater than 126 bytes, X25_WR_USER_DATA returns with an EINVAL error. This example copies CALL-user data from the array **udata** to the **userdata** structure, then writes the CALL-user data field:

(Requires Optional LAN/X.25 Software)

```
struct x25_userdata userdata;
unsigned char udata[128];
int i, j = 0, ndata;
ndata = num_bytes_CALL_user_data;
while (ndata > 0) {
        for (i = 0; ((ndata-- > 0) && ( i <= X25_MAX_CU_DATA)); i++)
                userdata.x25_cu_data[i] = udata[j++];
        userdata.x25_cud_len = i;
        error = ioctl(s, X25_WR_USER_DATA, &userdata);
        if (error)
                break;
}
```

X25_RD_FACILITIES

Reads any inbound facilities data received with an INCOMING CALL, CALL CONNECTED, CLEAR indication, or CLEAR confirm packet. Facilities are defined in Section 7 of the CCITT X.25 Recommendations. Facilities data cannot be reread, but the X25_RD_FACILITIES *ioctl ()* call can be issued again if new facilities data is received. X25_RD_FACILITIES cannot read outbound facilities data just written with the X25_WR_FACILITIES *ioctl ()* call.

X25_WR_FACILITIES

Sets up the facilities field. Facilities are defined in Section 7 of the CCITT X.25 Recommendations. The network provider might impose certain restrictions on facilities. If no facilities are specified before initiating a call, default flow-control facilities will be used. The programmer should ensure that the facilities written contain valid values and ranges as defined by the CCITT X.25 Recommendations and allowed by subscription by the network provider. If a user issues the X25_WR_FACILITIES *ioctl*() call with a field length of 0, no facilities are used for that connection. This call overwrites any outbound facilities data that hasn't been sent. Facilities data can also be overwritten by receiving a CALL packet or a CLEAR packet with a facilities field. The example below copies faciliies data from the **udata**[ ] array to the facilities data structure. Facilities data is stored in an **x25_facilities** data structure defined in **<x25/x25str.h>**:

```
struct x25_facilities fac_data;
unsigned char udata[128];
int ndata;
/* ndata" counts the number"
 *of bytes of facilities data.
 * udata[] contains the facilities data.
 */
memcpy(fac_data.x25_fac, udata, ndata);
fac_data.x25_fac_len = ndata;
error = ioctl(s, X25_WR_FACILITIES, &fac_data);
}
```

X25_RD_CTI

Returns the Circuit Table Index (CTI) associated with a particular socket. Knowing the CTI enables the user to read statistics and logging messages for a virtual circuit, as well as examine the contents of a network log file such as those created by *netlogstat*. The *ioctl*() call is issued as follows:

```
int error, s, cti;
error = ioctl (s, X25_RD_CTI, &cti);
```

X25_RD_CTI returns an error if the

X25_RD_LCI

Returns the Logical Channel Index (LCI) associated with a particular virtual circuit. Knowing the LCI enables the user to relate logging messages to a protocol-analyzer trace. The *ioctl*() call is issued as follows:

```
int error, s, lci; error = ioctl (s, X25_RD_LCI, &lci);
```

X25_RD_CTI returns an error if the circuit is not fully connected.

X25_WR_WTHRESHOLD
Sets the write threshold for a socket. This call is used with non-blocking I/O and *select*() calls. A socket is considered write-selectable if there is enough outbound buffer space for an X.25 message of the size specified with X25_WR_WTHRESHOLD. **Default:** 1 byte. **Range:** 1 byte to maximum *send*() message size specified with **setsockopt(SO_SNDBUF)**. The maximum outbound message size is 4096 bytes by default. Issue the X25_WR_WTHRESHOLD *ioctl ()* call as follows:

```
int s, wthresh, error;
wthresh = 1024;
error = ioctl( s, X25_WR_WTHRESHOLD, &wthresh);
```

X25_WR_CALLING_SUBADDR
Specifies a calling subaddress field to be included in the CALL REQUEST packet when initiating a connection request. This *ioctl*() is useful for applications where a client process initiates a virtual circuit and sends data to a server process which then shuts down the circuit; the server process manipulates the data, then reestablishes a virtual circuit with the same client (using a subaddress to identify it). The X25_WR_CALLING_SUBADDR *ioctl*() call enables the client to send addressing information to the server, which the server can retain after a connection is shut down. The addressing information is an X.121 sub-address. The sequence of system calls for executing the above procedure are summarized below:

1. The client loads X.121 sub-address information into an **x25addrstr** struct. Only the address family, subaddress, and size of subaddress fields in **x25addrstr** are used:

   ```
   #define SUBADDR = "12"
   struct x25addrstr addrstr;
   addrstr.x25_family = AF_CCITT;
   addrstr.x25_hostlen = strlen(SUBADDR);
   strncpy(addrstr.x25_host, SUBADDR, X25_MAXHOSTADDR);
   ```

2. The client binds to the address that the server will "call back", then listens on that socket:

   ```
   error = bind(s, &myaddr, sizeof(struct x25addrstr));
   error = listen(s, 5);
   ```

3. The client issues the X25_WR_CALLING_SUBADDR *ioctl*() call, with the subaddress information and all of the other fields of the structure stored in **addrstr**, then issues the *connect*() call:

   ```
   error = ioctl(s, X25_WR_CALLING_SUBADDR, &addrstr);
   error = connect(s, &to_addr, sizeof(struct x25addrstr));
   ```

4. The server receives the request, accepts the connection, issues *getpeername*() on the calling socket to obtain the addressing information, receives the data, then clears the connection.

   ```
   /* accept the connection request */
   new_s = accept(s, &from_addr, &from_addr_len);
   /* obtain addressing information for peer socket */
   error = getpeername(new_s, &from_addr_info, sizeof(struct x25addrstr));
   /* Clear the VC */
   error = close(new_s);
   ```

   For simplicity, no data exchanges between the client and server are shown, but presumably there will be one or more messages transmitted between them.

5. The server reestablishes a connection with the client process (which has been listening on the socket **s**) by creating a call-back socket, then issuing a *connect*() request using the subaddressing information and other X.25 addressing information from the old socket.

   ```
   call_back_s =
   socket(AF_CCITT, SOCK_STREAM, X25_PROTO_NUM)
   error = connect(call_back_s, &from_addr_info,
                   sizeof(struct x25addrstr));
   ```

X25_SET_FRAGMENT_SIZE
Sets fragment size for reception of an X.25 message in fragments. Normally, X.25/9000 software

sends data to the user in complete X.25 message sequences. If it is desired to receive X.25 normal data in fragments of the complete X.25 sequences, the X25_SET_FRAGMENT_SIZE I/O Control (see *ioctl*(2)) call must be used. This *ioctl*() informs the X.25/9000 software that inbound X.25 normal data should be sent to the user when the reassembled data size is equal to or greater than the size specified in the X25_SET_FRAGMENT_SIZE *ioctl*(). When the end of a complete X.25 message sequence is encountered, the remaining data is sent to the user.

The X25_SET_FRAGMENT_SIZE *ioctl*() can be used to set inbound fragment sizes up to 32767 bytes (32K). Values greater than 32K cause the *ioctl*() to return with an error indicating that the fragment size is too large, and the operation will be ignored. To return to receiving complete X.25 message sequences, set **fragment_size** to zero (0).

The semantics of the X25_SET_FRAGMENT_SIZE *ioctl*() is shown below:

```
int     socket, fragment_size;
extern int errno;

error=ioctl(socket, X25_SET_FRAGMENT_SIZE,
        &fragment_size);
if ((error == -1) && (errno == EMSGSIZE)) {
        printf( fragment_size too big0);"
}

if ((error == -1) && (errno == ENOTCONN)) {
        printf( circuit not connected0);"
}
```

The received fragment may be slightly larger than the **fragment_size** specified in the X25_SET_FRAGMENT_SIZE *ioctl*(). This is due to some inbound data reassembly being performed on the X.25 interface. It is highly recommended that the X25_NEXT_MSG_STAT *ioctl()* be used to determine the size of the message before it is received.

The received fragment may also be smaller than the **fragment_size specified in the** X25_SET_FRAGMENT_SIZE *ioctl*(). This is due to the detection of the end of a complete X.25 message sequence.

The X25_SET_FRAGMENT_SIZE *ioctl*() can be used only when the circuit is connected. It can be used at any time during the life of the circuit.

Inbound and outbound socket buffers should be set to at least the fragment size plus 4K (4096). This ensures that the proper amount of memory is reserved for operation of the virtual circuit. The default socket buffer size is 4K (4096 bytes).

DEPENDENCIES

Series 300/400 X.25:

Series 300/400 systems do not support setting the D-bit programmatically.

AUTHOR

socketx25 was developed by HP.

FILES

<x25/x25ioctls.h>
<x25/x25str.h>
<x25/x25addrstr.h>
<x25/x25com.h>

SEE ALSO

getsockopt(2), ioctl(2), socket(2), socket(7),

CCITT Data Communication Network Interfaces Recommendation X.25 ,

X.25 Programmer's Manual.

NAME

stty - terminal interface for Version 6/PWB compatibility

REMARKS

These facilities are included to aid in conversion of old programs, and should not be used in new code. Use the interface described in *termio*(7). Note that these conversions do *not* work for programs ported from UNIX Time-Sharing System, Seventh Edition (Version 7), because some V7 flags are defined differently.

DESCRIPTION

These routines attempt to map the UNIX Time-Sharing System, Sixth Edition (Version 6), and PWB *stty* and *gtty* calls into the current ioctls that perform the same functions. The mapping cannot be perfect. The way the features are translated is described below. The reader should be familiar with *termio*(7) before studying this entry.

The following data structure is defined in the include file <**sgtty.h**>:

```
struct sgttyb {
        char    sg_ispeed;      /* input speed */
        char    sg_ospeed;      /* output speed */
        char    sg_erase;       /* erase character */
        char    sg_kill;        /* kill character */
        int     sg_flags;       /* mode flags */
}
```

The flags, as defined in **sgtty.h**, are:

```
#define HUPCL     01
#define XTABS     02
#define LCASE     04
#define ECHO      010
#define CRMOD     020
#define RAW       040
#define ODDP      0100
#define EVENP     0200
#define ANYP      0300
#define NLDELAY   001400
#define TBDELAY   002000
#define CRDELAY   030000
#define VTDELAY   040000
#define BSDELAY   0100000

#define CR0       0
#define CR1       010000
#define CR2       020000
#define CR3       030000
#define NL0       0
#define NL1       000400
#define NL2       001000
#define NL3       001400
#define TAB0      0
#define TAB1      002000
#define NOAL      004000
#define FF0       0
#define FF1       040000
#define BS0       0
#define BS1       0100000
```

When the *stty*(2) command (*ioctl* **TIOCSETP**) is executed, the flags in the old **sgttyb** structure are mapped into their new equivalents in the **termio** structure. Then the **TCSETA** command is executed.

The following table shows the mapping between the old **sgttyb** flags and the current **termio** flags. Note that flags contained in the **termio** structure that are not mentioned below are cleared.

HUPCL	(if set) sets the **termio** HUPCL flag;
HUPCL	(if clear) clears the **termio** HUPCL flag;
XTABS	(if set) sets the **termio** TAB3 flag;
XTABS	(if clear) clears the **termio** TAB3 flag;
TBDELAY	(if set) sets the **termio** TAB1 flag;
TBDELAY	(if clear) clears the **termio** TAB1 flag;
LCASE	(if set) sets the **termio** IUCLC, OLCUC, and XCASE flags;
LCASE	(if clear) clears the **termio** IUCLC, OLCUC, and XCASE flags;
ECHO	(if set) sets the **termio** ECHO flag;
ECHO	(if clear) clears the **termio** ECHO flag;
NOAL	(if set) sets the **termio** ECHOK flag;
NOAL	(if clear) clears the **termio** ECHOK flag;
CRMOD	(if set) sets the **termio** ICRNL and ONLCR flags; also, if CR1 is set, the **termio** CR1 flag is set, and if CR2 is set, the **termio** ONOCR and CR2 flags are set;
CRMOD	(if clear) sets the **termio** ONLRET flag; also, if NL1 is set, the **termio** CR1 flag is set, and if NL2 is set, the **termio** CR2 flag is set;
RAW	(if set) sets the **termio** CS8 flag, and clears the **termio** ICRNL and IUCLC flags; also, default values of 6 characters and 0.1 seconds are assigned to MIN and TIME, respectively;
RAW	(if clear) sets the **termio** BRKINT, IGNPAR, ISTRIP, IXON, IXANY, OPOST, CS7, PARENB, ICANON, and ISIG flags; also, the default values control-D and null are assigned to the control characters EOF and EOL, respectively;
ODDP	(if set) if EVENP is also set, clears the **termio** INPCK flag; otherwise, sets the **termio** PARODD flag;
VTDELAY	(if set) sets the **termio** FFDLY flag;
VTDELAY	(if clear) clears the **termio** FFDLY flag;
BSDELAY	(if set) sets the **termio** BSDLY flag;
BSDELAY	(if clear) clears the **termio** BSDLY flag.

In addition, the **termio** CREAD bit is set, and, if the baud rate is 110, the CSTOPB bit is set.

When using **TIOCSETP**, the *ispeed* entry in the **sgttyb** structure is mapped into the appropriate speed in the **termio** CBAUD field. The *erase* and *kill* **sgttyb** entries are mapped into the **termio** erase and kill characters.

When the *gtty*(2) (*ioctl* **TIOCGETP**) command is executed, the *termio*(7) **TCGETA** command is first executed. The resulting **termio** structure is then mapped into the **sgttyb structure, which is then returned to the user.**

The following table shows how the **termio** flags are mapped into the old **sgttyb** structure. Note that all flags contained in the **sgttyb** structure that are not mentioned below are cleared.

HUPCL	(if set) sets the **sgttyb** HUPCL flag;
HUPCL	(if clear) clears the **sgttyb** HUPCL flag;
ICANON	(if set) sets the **sgttyb** RAW flag;
ICANON	(if clear) clears the **sgttyb** RAW flag;
XCASE	(if set) sets the **sgttyb** LCASE flag;
XCASE	(if clear) clears the **sgttyb** LCASE flag;
ECHO	(if set) sets the **sgttyb** ECHO flag;
ECHO	(if clear) clears the **sgttyb** ECHO flag;
ECHOK	(if set) sets the **sgttyb** NOAL flag;
ECHOK	(if clear) clears the **sgttyb** NOAL flag;
PARODD	(if set) sets the **sgttyb** ODDP flag;
PARODD	(if clear) clears the **sgttyb** ODDP flag;
INPCK	(if set) sets the **sgttyb** EVENP flag;
PARODD,	INPCK (if both clear) sets the **sgttyb** ODDP and EVENP flags;
ONLCR	(if set) sets the **sgttyb** CRMOD flag; also, if CR1 is set, the **sgttyb** CR1 flag is set, and if CR2 is set, the **sgttyb** CR2 flag is set;
ONLCR	(if clear) if CR1 is set, the **sgttyb** NL1 flag is set, and if CR2 is set, the **sgttyb** NL2 flag is set;
TAB3	(if set) sets the **sgttyb** XTABS flag;
TAB3	(if clear) clears the **sgttyb** XTABS flag;
TAB1	(if set) sets the **sgttyb** TBDELAY flag;

TAB1	(if clear) clears the **sgttyb** TBDELAY flag;
FFDLY	(if set) sets the **sgttyb** VTDELAY flag;
FFDLY	(if clear) clears the **sgttyb** VTDELAY flag;
BSDLY	(if set) sets the **sgttyb** BSDELAY flag;
BSDLY	(if clear) clears the **sgttyb** BSDELAY flag.

When using **TIOCGETP**, the **termio** CBAUD field is mapped into the *ispeed* and *ospeed* entries of the **sgttyb** structure. Also, the **termio** erase and kill characters are mapped into the *erase* and *kill* **sgttyb** entries.

Note that, since there is not a one-to-one mapping between the **sgttyb** and **termio** structures, unexpected results may occur when using the older **TIOCSETP** and **TIOCGETP** calls. Thus, the **TIOCSETP** and **TIOCGETP** calls should be replaced in all future code by the current equivalents, **TCSETA** and **TCGETA**, respectively.

SEE ALSO

stty(2), termio(7).

NAME

TCP - Internet Transmission Control Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, 0);
```

DESCRIPTION

The TCP protocol provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the **SOCK_STREAM** socket type. TCP constructs virtual circuits between peer entities. A virtual circuit consists of remote Internet addresses, remote ports, local Internet addresses and local ports. IP uses the Internet addresses to direct messages between hosts, and the port numbers to identify a TCP entity at a particular host.

Sockets using TCP are either **active** or **passive**. **connect()** creates active sockets, which initiate connections to passive sockets (see *connect*(2)). To create a passive socket, use the **listen()** system call after binding the socket with the **bind()** system call (see *listen*(2) and *bind*(2)). Only passive sockets can use the **accept()** call to accept incoming connections (see *accept*(2)).

Passive sockets can **underspecify** their location to match incoming connection requests from multiple networks. This technique, called **wildcard addressing**, allows a single server to provide service to clients on multiple networks. To create a socket that listens on all networks, the Internet address **INADDR_ANY** must be bound. The TCP port can still be specified even if wildcard addressing is being used. If the port is specified as zero, the system assigns a port.

Once **accept()** has a rendezvous with a connect request, a virtual circuit is established between peer entities. **bind()** supplies the local port and local Internet address and **accept()** gathers the remote port and remote Internet address from the peer requesting the connection.

The system supports one socket option, **TCP_NODELAY** (defined in the include file **<netinet/tcp.h>**), which is set with **setsockopt()** and tested with **getsockopt()**, and is explained below (see *getsockopt*(2)):

TCP_NODELAY	(boolean option; TCP SOCK_STREAM sockets only) causes small amounts of output to be sent immediately.

If **TCP_NODELAY** is set, the system sends small amounts of output immediately rather than gathering them into a single packet after an acknowledgement is received. If **TCP_NODELAY** is not set, the system sends data when it is presented, if there is no outstanding unacknowledged data. If there is outstanding unacknowledged data, the system gathers small amounts of data to be sent in a single packet once an acknowledgement is received. For clients such as window managers that send a stream of mouse events which receive no replies, this packetization may cause significant delays. The **TCP_NODELAY** option can be used to avoid this situation. Note, however, that setting the **TCP_NODELAY** option may result in a large number of small packets being sent over the network.

The default when a socket is created is that **TCP_NODELAY** is not set.

The option level to use for accessing the TCP option with the **setsockopt()** or **getsockopt()** calls is the protocol number for TCP which is available from **getprotobyname()** (see *getprotobyname*(3N)).

DIAGNOSTICS

One of the following errors may be returned if a socket operation fails. See the manual entry for the specific system call for a more complete and specific list of errors.

[EISCONN]	Attempt to establish a connection on a socket which already has one, or attempted operation on a connected socket that can only be done on one that has not been connected.
[ENOBUFS]	System ran out of memory for an internal data structure.
[ETIMEDOUT]	Connection dropped due to excessive retransmissions.
[ECONNRESET]	Remote peer forces the connection to be closed.
[ECONNREFUSED]	Remote peer actively refuses connection establishment (usually because no process is listening to the port).

[EADDRINUSE]	Attempt create a socket with a port that has already been allocated.
[EADDRNOTAVAIL]	Attempt to create a socket with a network address for which no network interface exists.

DEPENDENCIES

This entry describes the use of the TCP protocol as it applies to the ARPA Services Inter-Process Communication utility. It does not apply to the use of TCP for the LAN/9000 NetIPC utility on HP 9000 systems. Refer to the *HP 9000 NetIPC Programmer's Guide* for information about NetIPC.

If the **SO_KEEPALIVE** option is set on an established TCP connection, and the connection has been idle for two hours, TCP sends a packet to the remote socket, expecting the remote socket to acknowledge that it is still alive. If the remote socket does not respond within 75 seconds, TCP sends another packet. If TCP sends a total of 8 packets without response from the remote socket (that is, 10 minutes have passed), TCP drops the connection and the next socket call (e.g., **recv()**) returns an error with **errno** set to ETIMEDOUT. See *getsockopt*(2) for details on setting **SO_KEEPALIVE**.

The maximum buffer size for a TCP stream socket is 58254 bytes. The default buffer size is 8192 bytes. The send and receive buffer sizes for TCP stream sockets can be altered by using the **SO_SNDBUF** and **SO_RCVBUF** options of the **setsockopt()** system call. Refer to *getsockopt*(2) for details.

AUTHOR

TCP was developed by the University of California, Berkeley.

SEE ALSO

getsockopt(2), socket(2), inet(7F).

NAME

termio, termios - general terminal interface

DESCRIPTION

All HP-UX asynchronous communications ports use the same general interface, regardless of what hardware is involved. Network connections such as `rlogin` (see *rlogin*(1) use the pseudo-terminal interface (see *pty*(7).

This discussion centers around the common features of this interface.

Opening a Terminal File

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files; they are opened by special programs such as `getty` (see *getty*(1M)) and become a user's standard input, standard output, and standard error files.

If both the `O_NDELAY` and `O_NONBLOCK` flags (see *open*(2)) are clear, an *open* blocks until the type of modem connection requested (see *modem*(7)) is completed. If either the `O_NDELAY` or `O_NONBLOCK` flag is set, an *open* succeeds and return immediately without waiting for the requested modem connection to complete. The `CLOCAL` flag (see *Control Modes*) can also affect *open*(2).

Process Groups

A terminal can have a foreground process group associated with it. This foreground process group plays a special role in handling signal-generating input characters.

Command interpreter processes can allocate the terminal to different *jobs* (process groups) by placing related processes in a single process group and associating this process group with the terminal. A terminal's foreground process group can be set or examined by a process, assuming that the permission requirements are met (see *tcsetpgrp*(3C) or *tcgetpgrp*(3C)). The terminal interface aids in this allocation by restricting access to the terminal by processes that are not in the foreground process group.

A process group is considered orphaned when the parent of every member of the process group is either itself a member of the process group or is not a member of the group's session (see *Sessions*).

Sessions

A process that creates a session (see *setsid*(2) or *setpgrp*(2)) becomes a session leader. Every process group belongs to exactly one session. A process is considered to be a member of the session of which its process group is a member. A newly created process joins the session of its parent. A process can change its session membership (see *setpgid*(2) or *setpgrp2*(2)). Usually a session comprises all the processes (including children) created as a result of a single login.

The Controlling Terminal

A terminal can belong to a process as its controlling terminal. Each process of a session that has a controlling terminal has the same controlling terminal. A terminal can be the controlling terminal for at most one session. The controlling terminal for a session is allocated by the session leader. If a session leader has no controlling terminal and opens a terminal device file that is not already associated with a session without using the `O_NOCTTY` option (see *open*(2), the terminal becomes the controlling terminal of the session and the controlling terminal's foreground process group is set to the process group of the session leader. While a controlling terminal is associated with a session, the session leader is said to be the controlling process of the controlling terminal.

The controlling terminal is inherited by a child process during a `fork()` (see *fork*(2)). A process relinquishes its controlling terminal if it creates a new session with `setsid()` or `setpgrp()` (see *setsid*(2) and *setpgrp*(2)), or when all file descriptors associated with the controlling terminal have been closed.

When the controlling process terminates, the controlling terminal is disassociated from the current session, allowing it to be acquired by a new session leader. A `SIGHUP` signal is sent to all processes in the foreground process group of the controlling terminal. Subsequent access to the terminal by other processes in the earlier session can be denied (see *Terminal Access Control*) with attempts to access the terminal treated as if a modem disconnect had been sensed.

Terminal Access Control

Read operations are allowed (see *Input Processing and Reading Data*) from processes in the foreground process group of their controlling terminal. If a process is not in the foreground process group of its controlling terminal, the process and all member's of its process group are considered to be in a background process group of this controlling terminal. All attempts by a process in a background process group to read from its

controlling terminal will be denied. If denied and the reading process is ignoring or blocking the SIGTTIN signal, or the process (on systems that implement *vfork* separately from *fork*) has made a call to *vfork*(2) but has not yet made a call to *exec*(2), or the process group of the reading process is orphaned, **read()** returns –1 with **errno** set to EIO and no signal is sent. In all other cases where the read is denied, the process group of the reading process will be sent a **SIGTTIN** signal. The default action of the **SIGTTIN** signal is to stop the process to which it is sent.

If the process is in the foreground process group of its controlling terminal, write operations are allowed (see *Writing Data and Output Processing*). Attempts by a process in a background process group to write to its controlling terminal are denied if **TOSTOP** (see *Local Modes*) is set, the process is not ignoring and not blocking the **SIGTTOU** signal, and the process (on systems that implement *vfork* separately from *fork*) has not made a call to *vfork*(2) without making a subsequent call to *exec*(2). If the write is denied and the background process group is orphaned, the **write()** returns –1 with **errno** set to **EIO**. If the write is denied and the background process group is not orphaned, the **SIGTTOU** signal is sent to the process group of the writing process. The default action of the **SIGTTOU** signal is to stop the process to which it is sent.

Certain calls that set terminal parameters are treated in the same fashion as write, except that **TOSTOP** is ignored; that is, the effect is identical to that of terminal writes when **TOSTOP** is set.

Input Processing and Reading Data

A terminal device associated with a terminal device file can operate in full-duplex mode, so that data can arrive, even while data output is occurring. Each terminal device file has an *input queue* associated with it into which incoming data is stored by the system before being read by a process. The system imposes a limit, **MAX_INPUT**, on the number of characters that can be stored in the input queue. This limit is dependent on the particular implementation, but is at least 256. When the input limit is reached, all saved characters are discarded without notice.

All input is processed either in canonical mode or non-canonical mode (see *Canonical Mode Input Processing* and *Non-Canonical Mode Input Processing*). Additionally, input characters are processed according to the **c_iflag** (see *Input Modes*) and **c_lflag** (see *Local Modes*) fields. For example, such processing can include *echoing*, which in general means transmitting input characters immediately back to the terminal when they are received from the terminal. This is useful for terminals that operate in full-duplex mode.

The manner in which data is provided to a process reading from a terminal device file depends on whether the terminal device file is in canonical or non-canonical mode.

Another dependency is whether the **O_NONBLOCK** or **O_NDELAY** flag is set by either *open*(2) or *fcntl*(2). If the **O_NONBLOCK** and **O_NDELAY** flags are both clear, the read request is blocked until data is available or a signal is received. If either the **O_NONBLOCK** or **O_NDELAY** flag is set, the read request completes without blocking in one of three ways:

- If there is enough data available to satisfy the entire request, **read()** completes successfully, having read all of the data requested, and returns the number of characters read.
- If there is not enough data available to satisfy the entire request, **read()** completes successfully, having read as much data as possible, and returns the number of characters read.
- If there is no data available, **read()** returns –1, with **errno** set to **EAGAIN** when the **O_NONBLOCK** flag is set. Otherwise, (flag **O_NONBLOCK** is clear and **O_NDELAY** is set) **read()** completes successfully, having read no data, and returns a count of 0.

The availability of data depends upon whether the input processing mode is canonical or non-canonical. The following sections, *Canonical Mode Input Processing* and *Non-Canonical Mode Input Processing*, describe each of these input processing modes.

Canonical Mode Input Processing (Erase and Kill Processing)

In canonical mode input processing, terminal input is processed in units of lines, where a line is delimited by a new-line (NL) character, an end-of-file (EOF) character, or an end-of-line character (EOL). See *Special Characters* for more information on **NL**, **EOF**, and **EOL**. This means that a read request does not return until an entire line has been typed or a signal has been received. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters can be requested in a read, even one, without losing information.

MAX_CANON is the limit on the number of characters in a line. This limit varies with each particular implementation, but is at least 256.

When the **MAX_CANON** limit is reached, all characters in the current undelimited line are discarded without notice.

Erase and kill processing occur when either of two special characters, the ERASE and KILL characters (see *Special Characters*), is received. This processing affects data in the input queue that has not yet been delimited by a NL, EOF, or EOL character. This undelimited data makes up the current line. The ERASE character deletes the last character in the current line, if one exists. The KILL character deletes all data in the current line, if any, and optionally outputs a new-line (NL) character. Both of these characters operate on a key-stroke basis, independent of any backspacing or tabbing that may have preceded them. ERASE and KILL characters have no effect if the the current line is empty. ERASE and KILL characters are not placed in the input queue.

Non-Canonical Mode Input Processing (MIN/TIME Interaction)

In non-canonical mode input processing, input characters are not assembled into lines, and erase and kill processing does not occur. The values of the **MIN** and **TIME** members of the **c_cc** array (see *termios Structure*) are used to determine how to process the characters received. **MIN** represents the minimum number of characters that should be received before **read()** successfully returns. TIME is a timer of 0.10 second granularity that is used to timeout bursty and short term data transmissions. The four possible cases for MIN and TIME and their interactions are described below.

Case A: MIN > 0, TIME > 0

In this case, TIME serves as an inter-character timer and is activated after the first character is received. Since it is an inter-character timer, it is reset after each character is received. The interaction between MIN and TIME is as follows:

- As soon as one character is received, the inter-character timer is started.
- If MIN characters are received before the inter-character timer expires (remember that the timer is reset upon receipt of each character), the read is satisfied. If the timer expires before MIN characters are received, the characters received to that point are returned to the user.
- Note that if TIME expires, at least one character will be returned because the timer would not have been enabled unless a character was received. In this case (MIN > 0, TIME > 0) the read blocks until the MIN and TIME mechanisms are activated by the receipt of the first character, or a signal is received.

Case B: MIN > 0, TIME = 0

In this case, since the value of TIME is zero, the timer plays no role and only MIN is significant. A pending read is not satisfied until MIN characters are received after any previous read completes (that is, the pending read blocks until MIN characters are received), or a signal is received. A program that uses this case to handle record-based terminal I/O can block indefinitely in the read operation.

Case C: MIN = 0, TIME > 0

In this case, since the value of MIN is zero, TIME no longer represents an inter-character timer. It now serves as a read timer that is activated as soon as the **read()** function is processed. A read is satisfied as soon as a single character is received *or* the read timer expires. If the timer expires, no character is returned. If the timer does not expire, the only way the read can be satisfied is by a character being received. A read cannot block indefinitely waiting for a character because if no character is received within TIME × 0.10 seconds after the read is initiated, **read()** returns a value of zero, having read no data.

Case D: MIN = 0, TIME = 0

The number of characters requested or the number of characters currently available, whichever is less, is returned without waiting for more characters to be input. If no characters are available, **read()** returns a value of zero, having read no data.

Some points to note about MIN and TIME:

1. In the above explanations, the interactions of MIN and TIME are not symmetric. For example, when MIN > 0 and TIME = 0, TIME has no effect. However, in the opposite case where MIN = 0 and TIME > 0, both MIN and TIME play a role in that MIN is satisfied with the receipt of a single

character.

2. Also note that in case A (MIN > 0, TIME > 0), TIME represents an inter-character timer while in case C (MIN = 0, TIME > 0), TIME represents a read timer.

These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B (where MIN > 0) exist to handle burst mode activity (such as file transfer programs) where a program would like to process at least MIN characters at a time. In case A, the inter-character timer is activated by a user as a safety measure while in case B it is turned off.

Cases C and D exist to handle single character timed transfers. These cases are readily adaptable to screen-based applications that need to know if a character is present in the input queue before refreshing the screen. In case C the read is timed, while in case D it is not.

Another important note is that MIN is always just a minimum. It does not denote a record length. For example, if a program initiates a read of 20 characters when MIN is 10 and 25 characters are present, 20 characters will be returned to the user. Had the program requested all characters, all 25 characters would be returned to the user.

Furthermore, if TIME is greater than zero and MIN is greater than **MAX_INPUT**, the read will never terminate as a result of MIN characters being received because all the saved characters are discarded without notice when **MAX_INPUT** is exceeded. If TIME is zero and MIN is greater than **MAX_INPUT**, the read will never terminate unless a signal is received.

Special Characters

Certain characters have special functions on input, output, or both. Unless specifically denied, each special character can be changed or disabled. To disable a character, set its value to **_POSIX_VDISABLE** (see *unistd*(5)). These special functions and their default character values are:

INTR — (Rubout or ASCII DEL) special character on input and is recognized if **ISIG** (see *Local Modes*) is enabled. Generates a **SIGINT** signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. Normally, each such process is forced to terminate, but arrangements can be made to either ignore or hold the signal, or to receive a trap to an agreed-upon location; see *signal*(2) and *signal*(5). If **ISIG** is set, the INTR character is discarded when processed. If **ISIG** is clear, the INTR character is processed as a normal data character, and no signal is sent.

QUIT — (Ctrl-| or ASCII FS) special character on input. Recognized if **ISIG** (see *Local Modes*) is set. The treatment of this character is identical to that of the INTR character except that a **SIGQUIT** signal is generated and the processes that receive this signal are not only terminated, but a core image file (called **core**) is created in the current working directory if the implementation supports core files.

SWTCH — (ASCII NUL) special character on input and is only used by the shell layers facility *shl*(1). The shell layers facility is not part of the general terminal interface. No special functions are performed by the general terminal interface when SWTCH characters are encountered.

ERASE — (#) special character on input and is recognized if **ICANON** (see *Local Modes*) is enabled. Erases the preceding character. Does not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character. If **ICANON** is enabled, the ERASE character is discarded when processed. If **ICANON** is not enabled, the ERASE character is treated as a normal data character.

KILL — (@) special character on input and is recognized if **ICANON** is enabled. KILL deletes the entire line, as delimited by a NL, EOF, or EOL character. If **ICANON** is enabled, the KILL character is discarded when processed. If **ICANON** is not enabled, the KILL character is treated as a normal data character.

EOF — (Control-D or ASCII EOT) special character on input and is recognized if **ICANON** is enabled. EOF can be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, (that is, the EOF occurred at the beginning of a line) a character count of zero is returned from **read()**, representing an end-of-file indication. If **ICANON** is enabled,

the EOF character is discarded when processed. If **ICANON** is not enabled, the EOF character is treated as a normal data character.

NL (ASCII LF) special character on input and is recognized if **ICANON** flag is enabled. It is the line delimiter (\n). If **ICANON** is not enabled, the NL character is treated as a normal data character.

EOL (ASCII NUL) special character on input and is recognized if **ICANON** is enabled. EOL is an additional line delimiter similar to NL. It is not normally used. If **ICANON** is not enabled, the EOL character is treated as a normal data character.

SUSP (disabled) special character recognized on input. If **ISIG** is enabled, receipt of the SUSP character causes a **SIGTSTP** signal to be sent to all processes in the foreground process group for which the terminal is the controlling terminal, and the SUSP character is discarded when processed. If **ISIG** is not enabled, the SUSP character is treated as a normal data character. Command interpreter processes typically set SUSP to Control-Z.

STOP (Control-S or ASCII DC3) special character on both input and output. If **IXON** (output control) is enabled, processing of the STOP character temporarily suspends output to the terminal device. This is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended and **IXON** is enabled, STOP characters are ignored and not read. If **IXON** is enabled, the STOP character is discarded when processed. If **IXON** is not enabled, the STOP character is treated as a normal data character. If **IXOFF** (input control) is enabled, the system sends a STOP character to the terminal device when the number of unread characters in the input queue is approaching a system specified limit. This is an attempt to prevent this buffer from overflowing by telling the terminal device to stop sending data.

START (Control-Q or ASCII DC1) special character on both input and output. If **IXON** (output control) is enabled, processing of the START character resumes output that has been suspended. While output is not suspended and **IXON** is enabled, START characters are ignored and not read. If **IXON** is enabled, the START character is discarded when processed. If **IXON** is not enabled, the START character is treated as a normal data character. If IXOFF (input control) is enabled, the system sends a START character to the terminal device when the input queue has drained to a certain system-defined level. This occurs when the input queue is no longer in danger of possibly overflowing.

CR (ASCII CR) special character on input is recognized if **ICANON** is enabled. When **ICANON** and **ICRNL** are enabled and **IGNCR** is not enabled, this character is translated into a NL, and has the same affect as the NL character. If **ICANON** and **IGNCR** are enabled, the CR character is ignored. If **ICANON** is enabled and both **ICRNL** and **IGNCR** are not enabled, the STOP character is treated as a normal data character.

The NL, CR, START, and STOP characters cannot be changed or disabled. The character values for INTR, QUIT, ERASE, KILL, EOF, SWTCH, SUSP, and EOL can be changed or disabled to suit individual tastes. If **ICANON** is set (see *Local Modes*), the ERASE, KILL, and EOF characters can be escaped by a preceding \ character, in which case no special function is performed.

If two or more special characters have the same value, the function performed when the character is processed is undefined.

Modem Disconnect

If a modem disconnect is detected by the terminal interface for a controlling terminal, and if **CLOCAL** is clear in the **c_cflag** field for the terminal (see *Control Modes*), the **SIGHUP** signal is sent to the controlling process of the controlling terminal. Unless other arrangements have been made, this causes the controlling process to terminate. Any subsequent read from the terminal device returns with an end-of-file indication until the device is closed. Thus, processes that read a terminal file and test for end-of-file can terminate appropriately after a disconnect. Any subsequent **write()** to the terminal device returns –1, with **errno** set to EIO, until the device is closed.

Closing a Terminal Device File

The last process to close a terminal device file causes any output not already sent to the device to be sent to the device even if output was suspended. This last close always blocks (even if non-blocking I/O has been

specified) until all output has been sent to the terminal device. Any input that has been received but not read is discarded.

Writing Data and Output Processing

When characters are written, they are placed on the output queue. Characters on the output queue are transmitted to the terminal as soon as previously-written characters are sent. These characters are processed according to the `c_oflag` field (see *Output Modes*). Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters for output more rapidly than they can be sent, the process is suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the process is resumed.

termios Structure

Routines that need to control certain terminal I/O characteristics can do so by using the `termios` structure as defined in the header file <`termios.h`>. The structure is defined as follows:

```
#define NCCS    16
struct  termios {
        tcflag_t   c_iflag;     /* input modes */
        tcflag_t   c_oflag;     /* output modes */
        tcflag_t   c_cflag;     /* control modes */
        tcflag_t   c_lflag;     /* local modes */
        tcflag_t   c_reserved;  /* reserved for future use */
        cc_t       c_cc[NCCS];  /* control chars */
};
```

The special characters are defined by the array `c_cc`. The relative positions and initial values for each special character function are as follows:

EOF	VEOF	Control-D
EOL	VEOL	NUL
ERASE	VERASE	#
INTR	VINTR	DEL
KILL	VKILL	@
MIN	VMIN	NUL
QUIT	VQUIT	Control-\
START	VSTART	Control-Q
STOP	VSTOP	Control-S
SUSP	VSUSP	disabled
SWTCH	VSWTCH	NUL
TIME	VTIME	Control-D

termio Structure

The `termio` structure has been superseded by the `termios` structure and is provided for backward compatibility with prior applications (see *termio Caveats*). The structure is defined in the header file <`termio.h`> and is defined as follows:

```
#define NCC 8
struct  termio {
    unsigned short  c_iflag;     /* input modes */
    unsigned short  c_oflag;     /* output modes */
    unsigned short  c_cflag;     /* control modes */
    unsigned short  c_lflag;     /* local modes */
    char            c_line;      /* line discipline */
    unsigned char   c_cc[NCC];   /* control chars */
};
```

Modes

The next four sections describe the specific terminal characteristics that can be set using the `termios` and `termio` structures (see *termio Caveats*). Any bits in the modes fields that are not explicitly defined below are ignored. However, they should always be clear to prevent future compatibility problems.

Input Modes

The `c_iflag` field describes the basic terminal input control:

IGNBRK	Ignore break condition.
BRKINT	Signal interrupt on break.
IGNPAR	Ignore characters with parity errors.
PARMRK	Mark parity errors.
INPCK	Enable input parity check.
ISTRIP	Strip character.
INLCR	Map NL to CR on input.
IGNCR	Ignore CR.
ICRNL	Map CR to NL on input.
IUCLC	Map uppercase to lowercase on input.
IXON	Enable start/stop output control.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.

A break condition is defined as a sequence of zero-value bits that continues for more than the time to send one character. For example, a character framing or parity error with data all zeros is interpreted as a single break condition.

If **IGNBRK** is set, the break condition is ignored. Therefore the break condition cannot be read by any process. If **IGNBRK** is clear and **BRKINT** is set, the break condition flushes both the input and output queues and, if the terminal is the controlling terminal of a foreground process group, the break condition generates a single **SIGINT** signal to that foreground process group. If neither **IGNBRK** nor **BRKINT** is set, a break condition is read as a single **\0** character, or if **PARMRK** is set, as the three-character sequence **\377**, **\0**, **\0**.

If **IGNPAR** is set, characters with other framing and parity errors (other than break) are ignored.

If **PARMRK** is set, and **IGNPAR** is clear, a character with a framing or parity error (other than break) is read as the three-character sequence: **\377**, **\0**, *X*, where *X* is the data of the character received in error. To avoid ambiguity in this case, if **ISTRIP** is clear, a valid character of **\377** is read as **\377**, **\377**. If both **PARMRK** and **IGNPAR** are clear, a framing or parity error (other than break) is read as the character **\0**.

If **INPCK** is set, input parity checking is enabled. If **INPCK** is clear, input parity checking is disabled. Whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled (see *Control Modes*). If **PARENB** is set (see *Control Modes*) and **INPCK** is clear, parity generation is enabled but input parity checking is disabled; the hardware to which the terminal is connected will recognize the parity bit, but the terminal special file will not check whether this bit is set correctly or not.

The following table shows the interrelationship between the flags **IGNBRK**, **BRKINT**, **IGNPAR**, and **PARMRK**. The column marked **Input** gives various types of input characters received, indicated as follows:

0	NUL character (**\0**)
C	Character other than NUL
P	Parity error detected
F	Framing error detected

Items enclosed in brackets indicate one or more of the conditions are true.

If the **INPCK** flag is clear, characters received with parity errors are not processed according to this table, but instead, as if no parity error had occurred. Under the flag columns, **Set** indicates the flag is set, **Clear** indicates the flag is not set, and **X** indicates the flag may be set or clear. The column labeled **Read** shows the results that will be passed to the application code. A — indicates that no character or condition is passed to the application code. The value **SIGINT** indicates that no character is returned, but that the **SIGINT** signal is sent to the foreground process group of the controlling terminal.

Input	IGNBRK	BRKINT	IGNPAR	PARMRK	Read
0[PF]	Set	X	X	X	—
0[PF]	Clear	Set	X	X	**SIGINT**
0[PF]	Clear	Clear	X	Set	'\377','\0','\0'
0[PF]	Clear	Clear	X	Clear	'\0'
C[PF]	X	X	Set	X	—
C[PF]	X	X	Clear	Set	'\377','\0',C
C[PF]	X	X	Clear	Clear	'\0'

'\377'	X	X	X	Set	'\377','\377'

If **ISTRIP** is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If **INLCR** is set, a received NL character is translated into a CR character. If **IGNCR** is set, a received CR character is ignored (not read). If **IGNCR** is clear and **ICRNL** is set, a received CR character is translated into a NL character.

If **IUCLC** and **IEXTEN** are set, a received uppercase alphabetic character is translated into the corresponding lowercase character.

If **IXON** is set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. If **IXANY**, **IXON**, and **IEXTEN** are all set, any input character without a framing or parity error restarts output that has been suspended. When these three flags are set, output suspended, and an input character received with a framing or parity error, output resumes if processing it results in data being read. When **IXON** is set, START and STOP characters are not read, but merely perform flow control functions. When **IXON** is clear, the START and STOP characters are read.

If **IXOFF** is set, start/stop input control is enabled. The system transmits a STOP character when the number of characters in the input queue exceeds a system defined value (high water mark). This is intended to cause the terminal device to stop transmitting data in order to prevent the number of characters in the input queue from exceeding **MAX_INPUT**. When enough characters have been read from the input queue that the number of characters remaining is less than another system defined value (low water mark), the system transmits a START character which is intended to cause the terminal device to resume transmitting data (without risk of overflowing the input queue). In order to avoid potential deadlock, **IXOFF** is ignored in canonical mode whenever there is no line delimiter in the input buffer. In this case, the STOP character is not sent at the high water mark, but will be transmitted later if a delimiter is received. If all complete lines are read from the input queue leaving only a partial line with no line delimiter, the START character is sent, even if the number of characters is still greater than the low water mark. When **ICANON** is set and the input stream contains more characters between line delimiters than the high water mark allows, there is no guarantee that **IXOFF** can prevent buffer overflow and data loss, because the STOP character may not be sent in time, if at all.

The initial input control value is all bits clear.

Output Modes

The **c_oflag** field specifies the system treatment of output:

OPOST	Postprocess output.
OLCUC	Map lowercase to uppercase on output.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select new-line delays:
NL0	No delay
NL1	Delay type 1
CRDLY	Select carriage-return delays:
CR0	No delay
CR1	Delay type 1
CR2	Delay type 2
CR3	Delay type 3
TABDLY	Select horizontal-tab delays:
TAB0	No delay
TAB1	Delay type 1
TAB2	Delay type 2
TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	No delay
BS1	Delay type 1

VTDLY	Select vertical-tab delays:
VT0	No delay
VT1	Delay type 1
FFDLY	Select form-feed delays:
FF0	No delay
FF1	Delay type 1

If OPOST is set, output characters are post-processed as indicated by the remaining flags; otherwise characters are transmitted without change.

If OLCUC is set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0, and the delays specified for CR will be used. If ONLRET is clear, the NL character is assumed to perform only the line-feed function; the delays specified for NL are used and the column pointer remains unchanged. For all of these cases, the column pointer is always set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. The values of NL0, CR0, TAB0, BS0, VT0, and FF0 indicate no delay. If OFILL is set, fill characters are transmitted for delay instead of a timed delay. This is useful for high baud rate terminals, that need only a minimal delay. If OFDEL is set, the fill character is DEL; otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters are transmitted.

Carriage-return delay type 1 depends on the current column position; type 2 is about 0.10 seconds; type 3 about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters; type 2, four fill characters.

Horizontal-tab delay type 1 is depends on the current column position. Type 2 is about 0.10 seconds; type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters are transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character is transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

Control Modes

The c_cflag field describes the hardware control of the terminal:

CBAUD	Baud rate:	CSIZE	Character size:
B0	Hang up	CS5	5 bits
B50	50 baud	CS6	6 bits
B75	75 baud	CS7	7 bits
B110	110 baud	CS8	8 bits
B134	134.5 baud		
B150	150 baud	CSTOPB	Send two stop bits, else one.
B200	200 baud	CREAD	Enable receiver.
B300	300 baud	PARENB	Parity enable.
B600	600 baud	PARODD	Odd parity, else even.
B900	900 baud	HUPCL	Hang up on last close.
B1200	1200 baud	CLOCAL	Local line, else dial-up.
B1800	1800 baud	LOBLK	Reserved for use by *shl*(1).
B2400	2400 baud		
B3600	3600 baud		
B4800	4800 baud		
B7200	7200 baud		
B9600	9600 baud		

B19200	19200 baud
B38400	38400 baud
EXTA	External A
EXTB	External B

The CBAUD bits specify the baud rate. The zero baud rate, **B0**, is used to hang up the connection. If **B0** is specified, the modem control lines (see *modem*(7)) cease to be asserted. Normally, this disconnects the line. For any particular hardware, impossible speed changes are ignored. **CBAUD** is provided for use with the **termio** structure. When the **termios** structure is used, several routines are available for setting and getting the input and output baud rates (see *termios Structure Related Functions*).

The **CSIZE** bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If **CSTOPB** is set, two stop bits are used; otherwise one stop bit. For example, at 110 baud, many devices require two stop bits.

If **PARENB** is set, parity generation is enabled (a parity bit is added to each output character). Furthermore, parity detection is enabled (incoming characters are checked for the correct parity). If **PARENB** is set, **PARODD** specifies odd parity if set; otherwise even parity is used. If **PARENB** is clear, both parity generation and parity checking are disabled.

If **CREAD** is set, the receiver is enabled. Otherwise no characters can be received.

The specific effects of the **HUPCL** and **CLOCAL** bits depend on the mode and type of the modem control in effect. See *modem*(7) for the details.

If **HUPCL** is set, the modem control lines for the port are lowered (disconnected) when the last process using the open port closes it or terminates.

If **CLOCAL** is set, a connection does not depend on the state of the modem status lines. If **CLOCAL** is clear, the modem status lines are monitored.

Under normal circumstances, a call to **read()** waits for a modem connection to complete. However, if either the **O_NDELAY** or the **O_NONBLOCK** flags are set or **CLOCAL** is set, the **open()** returns immediately without waiting for the connection. If **CLOCAL** is set, see *Modem Disconnect* for the effects of **read()** and **write()** for those files for which the connection has not been established or has been lost.

LOBLK is used by the shell layers facility (see *shl*(1)). The shell layers facility is not part of the general terminal interface, and the **LOBLK** bit is not examined by the general terminal interface.

The initial hardware control value after open is **B300**, **CS8**, **CREAD**, and **HUPCL**.

Local Modes

The **c_lflag** field is used to control terminal functions.

ISIG	Enable signals.
ICANON	Canonical input (erase and kill processing).
XCASE	Canonical upper/lower presentation.
ECHO	Enable echo.
ECHOE	Echo ERASE as correcting backspace sequence.
ECHOK	Echo NL after kill character.
ECHONL	Echo NL.
NOFLSH	Disable flush after interrupt, quit, or suspend.
TOSTOP	Send SIGTTOU for background output.
IEXTEN	Enable extended functions.

If **ISIG** is set, each input character is checked against the special control characters INTR, QUIT, SUSP, and DSUSP (see *Process Group Control IOCTL Commands*). If an input character matches one of these control characters, the function associated with that character is performed and the character is discarded. If **ISIG** is clear, no checking is done and the character is treated as a normal data character. Thus these special input functions are possible only if **ISIG** is set.

If **ICANON** is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, or EOL. If **ICANON** is clear, read requests are satisfied directly from the input queue. A read blocks until at least MIN characters have been received or the timeout value TIME has expired between characters. (See *Non-Canonical Mode Input Processing*

(MIN/TIME Interaction)). This allows fast bursts of input to be read efficiently while still allowing single-character input. The time value represents tenths of seconds.

If **XCASE** is set, and if **ICANON** and **IEXTEN** are set, an uppercase letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

To obtain:	Use:
`	\'
|	\!
{	\(
}	\)
\	\\

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**. **XCASE** would normally be used in conjunction with **IUCLC** and **OLCUC** for terminals that support only the first-sixty-four-character limited character set. In this case, **IUCLC** processing is done before **XCASE** for input, and processing is done after **XCASE** for output. Therefore typing **A** causes an **a** to be read because of **IUCLC**, and typing **\A** causes an **A** to be read since **IUCLC** produces **\a** which is turned into **A** by the **XCASE** processing.

If **ECHO** is set, characters are echoed back to the terminal when received. If **ECHO** is clear, characters are not echoed.

When **ICANON** is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as described in *Canonical Mode Input Processing*. Furthermore, the following echo functions are possible. If **ECHO** and **ECHOE** are set, the erase character is echoed as the three-character ASCII sequence BS SP BS, which clears the last character from a CRT screen. If **ECHOE** is set and **ECHO** is clear, the erase character is echoed as the two-character ASCII sequence SP BS, which clears the current character from a CRT screen (the cursor remains in the same position). If **ECHOK** is set, the NL character is echoed after the kill character to emphasize that the line is being deleted. If **ECHONL** is set, the NL character is echoed even if **ECHO** is clear. This is useful for terminals set to local echo (that is, half duplex). Unless escaped, the EOF character is not echoed. Because ASCII EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If **NOFLSH** is set, the normal flush of the input and output queues associated with quit, interrupt, and suspend characters is not done. However, **NOFLSH** does not affect the flushing of data upon receipt of a break when **BRKINT** is set.

If the **TOSTOP** bit is set, an attempt by a process that is not in the foreground process group to write to its controlling terminal will be denied when the process is not ignoring and not blocking the **SIGTTOU** signal. If the write is denied and the process is a member of an orphaned process group **write()** returns –1 and sets **errno** to EIO and no signal is sent. If the write is denied and the process is a not a member of an orphaned process group, the **SIGTTOU** signal is sent to that process group.

If **ICANON** is set, the ERASE, KILL, and EOF characters can be escaped by a preceding \ character, in which case no special function is done.

If **IEXTEN** is set, **IXANY**, **XCASE**, and **IUCLC** functions are allowed. **IEXTEN** does not affect any other functions.

The initial local control value is all-bits-clear.

Special Control Characters

Special control characters are defined in the array **c_cc**. All of these special characters except START and STOP can be changed. Attempts to change the START and STOP are ignored. The subscript name and description for each element in both canonical and non-canonical mode are shown in the following table.

Subscript Usage

Canonical		Non-Canonical
VEOF		EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR	VINTR	INTR character
VKILL		KILL character

	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value

termios Structure-Related Functions

The following functions are provided when using the *termios* structure. Note that the effects on the terminal device do not become effective until the `tcsetattr()` function is successfully called. Refer to the appropriate manual entries for details.

termios Structure Functions

Function	Description
`cfgetospeed()`	get output baud rate
`cfgetispeed()`	get input baud rate
`cfsetospeed()`	set output baud rate
`cfsetispeed()`	set input baud rate
`tcgetattr()`	get terminal state
`tcsetattr()`	set terminal state

termio Structure-Related IOCTL Commands

Several `ioctl()` system calls apply to terminal files that use the `termio` structure (see *termio Structure*). If a requested command is not recognized, the request returns −1 with `errno` set to EINVAL.

`ioctl()` system calls that reference the `termio` structure have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

Commands using this form are:

TCGETA Get the parameters associated with the terminal and store them in the `termio` structure referenced by *arg*. This command is allowed from a background process; however, the information may be subsequently changed by a foreground process.

TCSETA Set the parameters associated with the terminal from the structure referenced by *arg*. The change is immediate. If characters are being output when the command is requested, results are undefined and the output may be garbled.

TCSETAW Wait for the output to drain before setting new parameters. This form should be used when changing parameters that affect output.

TCSETAF Wait for the output to drain, then flush the input queue and set the new parameters.

termio Caveats

Only the first eight special control characters (see *termios Structure*) can be set or returned. The values of indices VEOL and VEOF are the same as indices VTIME and VMIN respectively. Hence if **ICANON** is set, VEOL or VTIME is the additional end-of-line character and VEOF or VMIN is the end-of-file character. If **ICANON** is clear, VEOL or VTIME is the inter-character-timer value and VEOF or VMIN is the minimum number of characters desired for reads.

The **IEXTEN** flag (see *Local Modes*) cannot be changed directly by **TCSETA**, **TCSETAW**, or **TCSETAF**, nor can it be returned by **TCGETA**: This flag is always considered set after a successful **TCSETA**, **TCSETAF**, or **TCSETAW** command. This flag stays set and its function is performed until a call that uses the *termios* structure specifically clears the flag.

Structure-Independent Functions

The following functions which are independent of both the `termio` and `termios` structures are provided for controlling terminals. Refer to the appropriate manual entries for details.

Structure-Independent Functions

Function	Description
`tcsendbreak()`	send a break
`tcdrain()`	wait until output has drained
`tcflush()`	flush input or output queue or both

`tcflow()`	suspend or resume input or output
`tcgetpgrp()`	get foreground process group id
`tcsetpgrp()`	set foreground process group id

System Asynchronous I/O IOCTL Commands

The following `ioctl()` system calls provide for system asynchronous I/O and have the form:

```
ioctl (fildes, command, arg)
int *arg;
```

Commands using this form are:

`FIOSSAIOSTAT` If the integer referenced by *arg* is non-zero, system asynchronous I/O is enabled; that is, enable `SIGIO` to be sent to the process currently designated with `FIOSSAIOOWN` (see below) whenever the terminal device file status changes from "no read data available" to "read data available". If no process has been designated with `FIOSSAIOOWN`, enable `SIGIO` to be sent to the first process that opened the terminal device file.

If the designated process has exited, the `SIGIO` signal is not sent to any process.

If the integer referenced by *arg* is 0, system asynchronous I/O is disabled.

The default on open of a terminal device file is that system asynchronous I/O is disabled.

`FIOGSAIOSTAT` The integer referenced by *arg* is set to 1 if system asynchronous I/O is enabled. Otherwise, the integer referenced by *arg* is set to 0.

`FIOSSAIOOWN` Set the process ID that will receive the `SIGIO` signals due to system asynchronous I/O to the value of the integer referenced by *arg*. If no process can be found corresponding to that specified by the integer referenced by *arg*, the call returns −1 with `errno` set to ESRCH. A user with appropriate privileges can designate that any process receive the `SIGIO` signals. If the request is not made by a user with appropriate privileges and the calling process does not either designate that itself or another process whose real, saved, or effective user ID matches its real or effective user ID or the calling process does not designate a process that is a descendant of the calling process to receive the `SIGIO` signals, the call returns −1 with `errno` set to EPERM.

If the designated process subsequently exits, the `SIGIO` signal is not sent to any process.

The default on open of a terminal device file is that the process performing the first open is set to receive the `SIGIO` signals.

`FIOGSAIOOWN` The integer referenced by *arg* is set to the process ID designated to receive `SIGIO` signals.

Line Control IOCTL Commands

Several `ioctl()` system calls control input and output. Some of these calls have the form:

```
ioctl (fildes, command, arg)
int *arg;
```

Commands using this form are:

`TCSBRK` Wait for the output to drain. If *arg* is 0, send a break (zero bits for at least 0.25 seconds). The `tcsendbreak()` function performs the same function (see *tcsendbreak*(3C)).

`TCXONC` Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output; if 2, transmit a STOP character; if 3, transmit a START character. If any other value is given for *arg*, the call returns −1 with `errno` set to EINVAL. The `tcflow()` function performs the same functions (see *tcflow*(3C)).

TCFLSH If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues. If any other value is given for *arg*, the call returns –1 with **errno** set to EINVAL. The **tcflush()** function performs the same functions (see *tcflush*(3C)).

Sending a BREAK is accomplished by holding the data transmit line at a SPACE or logical zero condition for at least 0.25 seconds. During this interval, data can be sent to the device, but because of serial data interface limitations, the BREAK takes precedence over all data. Thus, all data sent to a device during a BREAK is lost. This includes system-generated XON/XOFF characters used for input flow control. Note also that a delay in transmission of the XOFF flow control character until after the BREAK is terminated could still result in data overflow because the flow control character may not be sent soon enough.

Other calls have the form:

```
ioctl (fildes, command, arg)
long *arg;
```

Commands using this form are:

FIONREAD Returns in the long integer referenced by *arg* the number of characters immediately readable from the terminal device file. This command is allowed from a background process; however, the data itself cannot be read from a background process.

Non-blocking I/O IOCTL Commands

Non-blocking I/O is easily provided via the **O_NONBLOCK** and **O_NDELAY** flags available in both *open*(2) and *fcntl*(2). The commands in this section are provided for backward compatibility with previously developed applications. **ioctl()** system calls that provide a style of non-blocking I/O different from **O_NONBLOCK** and **O_NDELAY** have the form:

```
ioctl (fildes, command, arg)
long *arg;
```

Commands using this form are:

FIOSNBIO If the integer referenced by *arg* is non-zero, **FIOSNBIO**-style non-blocking I/O is enabled; that is, subsequent reads and writes to the terminal device file are handled in a non-blocking manner (see below). If the integer referenced by *arg* is 0, **FIOSNBIO**-style non-blocking I/O is disabled.

For reads, **FIOSNBIO**-style non-blocking I/O prevents all read requests to that device file from blocking, whether the requests succeed or fail. Such a read request completes in one of three ways:

- If there is enough data available to satisfy the entire request, the read completes successfully, having read all of the data, and returns the number of characters read;
- If there is not enough data available to satisfy the entire request, the read completes successfully, having read as much data as possible, and returns the number of characters read;
- If there is no data available, the read returns –1 with **errno** set to EWOULDBLOCK.

For writes, **FIOSNBIO**-style non-blocking I/O prevents all write requests to that device file from blocking, whether the requests succeed or fail. Such a write request completes in one of three ways:

- If there is enough space available in the system to buffer all the data, the write completes successfully, having written out all of the data, and returns the number of characters written;
- If there is not enough space in the buffer to write out the entire request, the write completes successfully, having written as much data as possible, and returns the number of characters written;

- If there is no space in the buffer, the write returns −1 with `errno` set to EWOULDBLOCK.

To prohibit `FIOSNBIO`-style non-blocking I/O from interfering with the `O_NONBLOCK` and `O_NDELAY` flags (see *open*(2) and *fcntl*(2)), the functionality of `O_NONBLOCK` and `O_NDELAY` always supersedes the functionality of `FIOSNBIO`-style non-blocking I/O. This means that if either `O_NONBLOCK` or `O_NDELAY` is set, the driver performs read requests in accordance with the definition of `O_NDELAY` or `O_NONBLOCK`. When both `O_NONBLOCK` and `O_NDELAY` are clear, the definition of `FIOSNBIO`-style non-blocking I/O applies.

The default on open of a terminal device file is that `FIOSNBIO`-style non-blocking I/O is disabled.

`FIOGSNBIO` The integer referenced by *arg* is set to 1, if `FIOSNBIO`-style non-blocking I/O is enabled. Otherwise, the integer referenced by *arg* is set to 0.

Process Group Control IOCTL Commands

The process group control features described here (except for setting and getting the delayed stop process character) are easily implemented using the functions `tcgetattr()`, `tcsetattr()`, `tcgetpgrp()`, and `tcsetpgrp()` (see *tcgetattr*(2), *tcsetattr*(2), *tcgetpgrp*(3C), and *tcsetpgrp*(3C) respectively).

The following structure, used with process group control, is defined in <`bsdtty.h`>:

```
struct  ltchars {
    unsigned char   t_suspc;    /* stop process character */
    unsigned char   t_dsuspc;   /* delayed stop process character */
    unsigned char   t_rprntc;   /* reserved; must be '_POSIX_VDISABLE' */
    unsigned char   t_flushc;   /* reserved; must be '_POSIX_VDISABLE' */
    unsigned char   t_werasc;   /* reserved; must be '_POSIX_VDISABLE' */
    unsigned char   t_lnextc;   /* reserved; must be '_POSIX_VDISABLE' */
};
```

The initial value for all these characters is `_POSIX_VDISABLE`, which causes them to be disabled. The meaning for each character is as follows:

`t_suspc` Suspend the foreground process group. A *suspend* signal (`SIGTSTP`) is sent to all processes in the foreground process group. Normally, each process is forced to stop, but arrangements can be made to either ignore or block the signal, or to receive a trap to an agreed-upon location; see *signal*(2) and *signal*(5). When enabled, the typical value for this character is Control-Z or ASCII SUB. Setting or getting `t_suspc` is equivalent to setting or getting the SUSP special control character.

`t_dsuspc` Same as **t_suspc**, except that the *suspend* signal (`SIGTSTP`) is sent when a process reads the character, rather than when the character is typed. When enabled, the typical value for this character is Ctrl-Y or ASCII EM.

Attempts to set any of the reserved characters to a value other than `_POSIX_VDISABLE` cause `ioctl()` to return −1 with `errno` set to EINVAL with no change in value of the reserved character.

`ioctl()` system calls that use the above structure have the form:

```
ioctl (fildes, command, arg)
struct ltchars *arg;
```

Commands using this form are:

`TIOCGLTC` Get the process group control characters and store them in the *ltchars* structure referenced by *arg*. This command is allowed from a background process. However, the information may be subsequently changed by a foreground process.

`TIOCSLTC` Set the process group control characters from the structure referenced by *arg*.

Additional process group control `ioctl()` system calls have the form:

```
ioctl (fildes, command, arg)
unsigned int *arg;
```

Commands using this form are:

`TIOCGPGRP` Returns in the integer referenced by *arg* the foreground process group associated with the terminal. This command is allowed from a background process. However, the information may be subsequently changed by a foreground process. This feature is easily implemented using the `tcgetpgrp()` function (see *tcgetpgrp*(3C)).

If the `ioctl()` call fails, it returns −1 and sets `errno` to one of the following values:

[EBADF] *fildes* is not a valid file descriptor.

[ENOTTY] The file associated with *fildes* is not the controlling terminal, or the calling process does not have a controlling terminal.

[EACCES] The file associated with *fildes* is the controlling terminal of the calling process, however, there is no foreground process group defined for the controlling terminal.

Note: EACCES may not be returned in future releases. Behavior in cases where no foreground process group is defined for the controlling terminal may change in future versions of the POSIX standard. Portable applications, therefore, should not rely on this error condition.

`TIOCSPGRP` Sets the foreground process group associated with the terminal to the value referenced by *arg*. This feature is easily implemented using the `tcsetpgrp()` function (see *tcsetpgrp*(3C)).

If the `ioctl()` call fails, it returns −1 and sets `errno` to one of the following values:

[EBADF] *fildes* is not a valid file descriptor.

[EINVAL] The process ID referenced by *arg* is not a supported value.

[ENOTTY] The calling process does not have a controlling terminal, or the *fildes* is not the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.

[EPERM] The process ID referenced by *arg* is a supported value but does not match the process group ID of a process in the same session as the calling process.

`TIOCLGET` Get the process group control mode word and store it in the int referenced by *arg*. This command is allowed from a background process; however, the information may be subsequently changed by a foreground process.

`TIOCLSET` Set the process group control mode word to the value of the int referenced by *arg*.

`TIOCLBIS` Use the int referenced by *arg* as a mask of bits to set in the process group control mode word.

`TIOCLBIC` Use the int referenced by *arg* as a mask of bits to clear in the process group control mode word.

The following bit is defined in the process group control mode word:

`LTOSTOP` Send `SIGTTOU` for background writes.

Setting or clearing `LTOSTOP` is equivalent to setting or clearing the `TOSTOP` flag (see *Local Modes*). If `LTOSTOP` is set and a process is not in the foreground process group of its controlling terminal, a write by the process to its controlling terminal may be denied (see *Terminal Access Control*).

Terminal Size IOCTL Commands

The following `ioctl()` system calls are used to get and set terminal size information for the terminal referenced by *fildes*. These `ioctl()` system calls use the `winsize` structure to get and set the terminal size information. The `winsize` structure, defined in <`termios.h`>, has the following members :

```
unsigned short ws_row;        /* Rows, in characters        */
unsigned short ws_col;        /* Columns, in characters     */
unsigned short ws_xpixel;     /* Horizontal size, in pixels */
unsigned short ws_ypixel;     /* Vertical size, in pixels   */
```

The initial values for all elements of terminal size are zero. The values for terminal size are neither set nor used by the general terminal interface, and have no effect on the functionality of the general terminal interface. The values for terminal size are set and used only by applications that access them through the terminal-size `ioctl()` system calls (see *ioctl*(2)).

`ioctl()` system calls that use the above structure have the form:

```
ioctl (fildes, command, arg)
struct winsize *arg;
```

Commands using this form are:

TIOCGWINSZ Get the terminal size values and store them in the **winsize** structure referenced by *arg*. This command is allowed from a background process.

TIOCSWINSZ Set the terminal size values from the **winsize** structure referenced by *arg*. If any of the new values differ from previous values, a **SIGWINCH** signal is sent to all processes in the terminal's foreground process group.

WARNINGS

Various HP-UX implementations use non-serial interfaces that look like terminals (such as bit-mapped graphics displays) or "smart cards" that cannot implement the exact capabilities described above. Therefore, not all systems can exactly meet the standard stated above. Each implementation is required to state any deviations from the standard as part of its system-specific documentation.

FIOSSAIOSTAT is similar to BSD 4.2 **FIOASYNC**, with the addition of provisions for security.

FIOGSAIOSTAT is of HP origin, complements **FIOSSAIOSTAT**, and allows saving and restoring system asynchronous I/O TTY states for command interpreter processes.

FIOSSAIOOWN is similar to BSD 4.2 **FIOSETOWN**, with additional provisions for security.

FIOGSAIOOWN is similar to BSD **FIOGETOWN**. 4.2 Note also the difference that the BSD 4.2 version of this functionality used process groups, while the HP-UX version only uses processes.

FIOSNBIO is the same as BSD FIONBIO, 4.2 except that it does not interfere with the **O_NDELAY** or **O_NONBLOCK** `open()` and `fcntl()` flags.

FIOGNBIO is of HP origin, complements **FIOSNBIO**, and allows saving and restoring the **FIOSNBIO**-style non-blocking I/O TTY state for command interpreter processes.

DEPENDENCIES

Series 300/400

Data loss can occur with HP 98626/98644 serial interfaces and the built-in serial interface of the Model 318, 319, 320, 330, 332, R/332, 340, 350, 360, V/360, and 370 if the effective combined data rate for all installed serial interfaces exceeds 2400 baud (for example, two interfaces running at 1200 baud and a third at 300 baud is equivalent to 2700 baud combined).

HP 98626/98644 serial interfaces and the built-in Series 300 serial interfaces of the Models listed above do not support 38 400 baud.

HP 98642/98638 serial interfaces do not support 200 and 38 400 baud. The second and third (select code 5,6) serial interfaces of Series 400 machines support no bit rates above 19200. On the Model 425e, this also applies to the first (sc 9) port.

Built-in serial ports on the Series 400 machines (except Model 425e) support the following additional baud rate settings: 57 600, 115 200, 230 400, and 460 800 baud. An RS-232-to-RS-422 converter may be required to achieve practical cable lengths at these baud rates (because RS-232 only specifies up to 19 200 baud).

The **c_iflag** field parameter IXANY (enable any character to restart output) is not supported by the HP 98628A interface card.

Timed delays are not supported.

The HP 98628A interface does not support the following baud rates: 900, 7200, 38 400.

Built-in serial ports on Series 400s and 400t machines and Series 300 models 332, 345, 375, 380, 382, and R/382 have both RTS and CTS flow control capability as well as a configurable receive FIFO trigger level and transmit limit.

RTS/CTS hardware handshaking can be enabled through a bit in the device file minor number, through an `ioctl()` call (see *termiox*(7)), or through the `stty` command (see *stty*(1)).

The receive FIFO trigger level is configurable through two bits in the device file minor number. The receive FIFO trigger level is used to set the level at which a receive interrupt is generated to the system. Setting a smaller value for the receive FIFO trigger level enables the system to react more quickly to receipt of characters. However, using a smaller trigger level increases system overhead to process the additional interrupts. A higher receive FIFO trigger level reduces the system interrupt overhead for heavy inbound data traffic at the cost of less time for the system to read data from the hardware before receive FIFOs are overrun. When using RTS flow control, the receive FIFO trigger level also determines the point at which the hardware lowers RTS to protect the receive FIFO. Use of a higher receive FIFO trigger level also reduces XOFF flow control responsiveness because, under light inbound data flow conditions, receipt of the XOFF character by the system is slightly delayed. Choice of the appropriate receive FIFO trigger level should be based upon how the serial port is to be used. For most applications a receive FIFO trigger level of 8 (c3,c2 = 10) is suggested.

Two bits in the device file minor number specify the transmit limit, the number of characters which are successively loaded into the transmit FIFO. Setting a smaller transmit limit allows the transmitter to be more responsive to flow control either from receipt of an XOFF character or de-assertion of CTS at the cost of increased system interrupt overhead. Setting a larger transmit limit reduces interrupt overhead but is not as responsive to flow control since the remainder of the transmit FIFO can be transmitted even after the transmitter is flow controlled. When communicating with devices which have little tolerance for data receipt after flow control, one must choose the transmit limit appropriately.

Series 300/400 device file minor number:
Series 300/400 device file minor numbers take the form:

`0x`*ScAdCM*

where:

`Sc` = Two hexadecimal digits (8 bits) to indicate the select code of the serial hardware.

`Ad` = Two hexadecimal digits (8 bits) to indicate port address. (HP 98642 and HP 98638 only)

`C` = One hexadecimal digit (4 bits) for FIFO control on the enhanced serial ports mentioned above. Values for each bit are as follows:

Receive FIFO Trigger Level			Transmit Limit		
c_3	c_2	Level	c_1	c_0	Limit
0	0	1	0	0	1
0	1	4	0	1	4
1	0	8	1	0	8
1	1	14	1	1	12

`M` = One hexadecimal digit (4 bits) to determine hardware flow state and port access type. Values for each bit are as follows:

Bit	Value
m_3	RTS/CTS hardware flow control (0 = OFF, 1 = ON)
m_2	0 = modem, 1 = direct connect
m_1	0 = Simple protocol (U.S.), 1 = CCITT protocol (Europe)
m_0	0 = dial-in modem, 1 = dial-out modem

Series 700

Built-in serial ports on Series 700 machines support the following additional baud rate settings: 57 600, 115 200, 230 400, and 460 800 baud. An RS-232-to-RS-422 converter may be required to achieve practical cable lengths at these baud rates (because RS-232 only specifies up to 19 200 baud).

Timed delays are not supported.

Built-in serial ports on Series 700 systems have RTS and CTS flow control capability, configurable receive FIFO trigger levels, and a configurable transmit limit. RTS/CTS hardware handshaking can be enabled through a bit in the device file minor number, through an `ioctl()` call (see *termiox*(7)), or through the `stty` command (see *stty*(1)). The discussion of receive FIFO trigger levels and transmit limits in the Series 300/400 section above also applies to built-in serial ports on Series 700 systems.

Series 700 device file minor number:

Series 700 device file minor numbers take the form:

0x*ScF*0*CM*

where:

`Sc` = Two hexadecimal digits (8 bits) to indicate the select code of the serial hardware. This is always 0x20.

`F` = One hexadecimal digit (4 bits) to specify the function number for the port (4 for port A, 5 for port B).

`C` = One hexadecimal digit (4 bits) for FIFO control. Values for each bit are as follows (same as Series 300/400):

Receive FIFO Trigger Level			Transmit Limit		
c_3	c_2	Level	c_1	c_0	Limit
0	0	1	0	0	1
0	1	4	0	1	4
1	0	8	1	0	8
1	1	14	1	1	12

`M` = One hexadecimal digit (4 bits) to determine hardware flow state and port access type. Values for each bit are as follows:

Bit	Value
m_3	RTS/CTS hardware flow control (0 = OFF, 1 = ON)
m_2	0 = modem, 1 = direct connect
m_1	0 = Simple protocol (U.S.), 1 = CCITT protocol (Europe)
m_0	0 = dial-in modem, 1 = dial-out modem

Series 800

Timed output delays are not directly supported. If used, an appropriate number of fill characters (based on the current baud rate) is output. The total time to output the fill characters is at least as long as the time requested.

The system specified input flow control values are as follows: low water mark is 60, high water mark is 180, and maximum allowed input is 512.

The HP 98196A (formerly 27140A option 800) interface does not support the following hardware settings:

`CBAUD B200, B38400, EXTA, EXTB.`

The HP A1703-60003 and the HP 28639-60001 interfaces do not support baud rates above 9600. Furthermore, changing the following hardware settings on port 0 from the default (9600 baud, 8 bit characters, 1 stop bit, no parity) is not supported:

`CBAUD CSIZE, CSTOPB, PARENB, PARODD.`

The HP J2094A interface supports RTS and CTS flow control. The RTS/CTS hardware handshaking can be enabled through a bit in the device file minor number, through an `ioctl()` call (see *termiox*(7)), or

through the **stty** command (see *stty*(1)). The device file minor number bits have the following meanings:

Series 800 device file minor number:
Series 800 device file minor numbers take the form:

0x*MHLuAD*

where:

M = One hexadecimal digit (4 bits) for the port access type. Values for each bit are as follows:

Bit	Value
m_3	Must be zero (0)
m_2	0 = Simple protocol (U.S.), 1 = CCITT protocol (Europe)
$m_1 m_0$	00 = Direct 01 = Dial-out modem 10 = Dial-in modem 11 = Invalid

H = One hexadecimal digit (4 bits) to determine hardware flow control (HP J2094A only). Values for each bit are as follows (bits 3, 1, and 0 must be 0):

$m_3 m_2 m_1 m_0$	Value
0000	RTS/CTS hardware flow control OFF
0100	RTS/CTS hardware flow control ON

Lu = Two hexadecimal digits (8 bits) to indicate the logical unit of the serial interface.

ad = Two hexadecimal digits (8 bits) to indicate the port number of this device on the serial interface.

AUTHOR

termios was developed HP and the IEEE Computer Society.
termio was developed by HP, AT&T, and the University of California, Berkeley.

FILES

/dev/console
/dev/tty*

SEE ALSO

shl(1), stty(1), mknod(1M), fork(2), ioctl(2), setsid(2), signal(2), stty(2), setpgid(2), blmode(3C), cfspeed(3C), tccontrol(3C), tcattribute(3C), tcgetpgrp(3C), tcsetpgrp(3C) signal(5), unistd(5), sttyV6(7), tty(7), modem(7), termiox(7).

STANDARDS CONFORMANCE

termio: SVID2, XPG2

termios: AES, XPG3, XPG4, FIPS 151-2, POSIX.1

(HP-PB Only)

NAME

termiox - extended general terminal interface

SYNOPSIS

#include <sys/termiox.h>

ioctl (int *fildes,* **int** *request,* **struct termiox** * *arg)*

DESCRIPTION

The extended general terminal interface supplements the *termio*(7) general terminal interface by adding support for asynchronous hardware flow control and local implementations of additional asynchronous features. Some systems may not support all of these capabilities because of hardware or software limitations. Other systems may not permit certain functions to be disabled. In such cases, the appropriate bits are ignored. If the capabilities can be supported, the interface described here must be used.

Hardware Flow Control Modes

Hardware flow control supplements the termio **IXON**, **IXOFF**, and **IXANY** character flow control (see *termio*(7)). Character flow control occurs when one device controls the data transfer of another device by inserting control characters in the data stream between devices. Hardware flow control occurs when one device controls the data transfer of another device by using electrical control signals on wires (circuits) of the asynchronous interface. Character flow control and hardware flow control can be simultaneously set.

In asynchronous, full duplex applications, the use of the Electronics Industries Association's EIA-232-D Request To Send (RTS) and Clear To Send (CTS) circuits is the preferred method of hardware flow control.

The EIA-232-D standard specified only unidirectional hardware flow control where the Data Circuit-terminating Equipment or Data Communications Equipment (DCE) indicates to the Data Terminal Equipment (DTE) to stop transmitting data. The termiox interface allows both unidirectional and bidirectional hardware flow control; when bidirectional flow control is enabled, either the DCE or DTE can indicate to each other to stop transmitting data across the interface.

Clock Modes

Isochronous flow control and clock mode communication are not supported.

Terminal Parameters

Parameters that control the behavior of devices providing the termiox interface are specified by the **termiox** structure, defined in the **<sys/termiox.h>** header file. Several **ioctl()** system calls (see *ioctl*(5)) that fetch or change these parameters use the **termiox** structure which contains the following members:

```
unsigned short x_hflag;     /* hardware flow control modes */
unsigned short x_cflag;     /* clock modes */
unsigned short x_rflag;     /* reserved modes */
unsigned short x_sflag;     /* spare local modes */
```

The **x_hflag** field describes hardware flow control modes:

RTSXOFF	**0000001**	Enable RTS hardware flow control on input.
CTSXON	**0000002**	Enable CTS hardware flow control on input.

The RTS and CTS circuits are involved in establishing CCITT modem connections. Since RTS and CTS circuits are used both by CCITT modem connections and by hardware flow control, CCITT modem and hardware flow control cannot be simultaneously enabled.

Variations of different hardware flow control methods can be selected by setting the appropriate bits. For example, bidirectional RTS/CTS flow control is selected by setting both the RTSXOFF and CTSXON bits. Unidirectional CTS hardware flow control is selected by setting only the CTSXON bit.

If RTSXOFF is set, the Request to Send (RTS) circuit (line) is raised, and if the asynchronous port needs to have its input stopped, it lowers the Request to Send (RTS) line. If the RTS line is lowered, it is assumed that the connected device will stop its output until RTS is raised.

If CTSXON is set, output occurs only if the Clear To Send (CTS) circuit (line) is raised by the connected device. If the CTS line is lowered by the connected device, output is suspended until CTS is raised.

termiox Structure Related IOCTL Command

The **ioctl()** system calls that reference the **termiox** structure have the form:

```
ioctl (fildes, command, arg)
struct termiox *arg;
```

Commands using this form are:

TCGETX The argument is a pointer to a **termiox** structure. The current terminal parameters are fetched and stored into that structure.

TCSETX The argument is a pointer to a **termiox** structure. The current terminal parameters are set from the values stored in that structure. The change is immediate. Errors that can be returned include:

[EINVAL] The port does not support hardware flow control.

[ENOTTY] The file descriptor for this port is configured for CCITT mode access. Hardware flow control is not allowed on CCITT mode devices.

TCSETXW The argument is a pointer to a **termiox** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output. Errors that can be returned include:

[EINVAL] The port does not support hardware flow control.

[ENOTTY] The file descriptor for this port is configured for CCITT mode access. Hardware flow control is not allowed on CCITT mode devices.

TCSETXF The argument is a pointer to a **termiox** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded, then the change occurs. Errors that can be returned include:

[EINVAL] The port does not support hardware flow control.

[ENOTTY] The file descriptor for this port is configured for CCITT mode access. Hardware flow control is not allowed on CCITT mode devices.

AUTHOR

termiox was developed by HP and AT&T.

FILES

Files in or under **/dev/tty***.

SEE ALSO

ioctl(2), termio(7), modem(7).

NAME

tty - controlling terminal interface

DESCRIPTION

The file **/dev/tty** is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that need to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

FILES

/dev/tty
/dev/tty*

SEE ALSO

termio(7).

STANDARDS CONFORMANCE

tty: AES, SVID2, XPG2, XPG3, XPG4, FIPS 151-2, POSIX.1

NAME

udp - internet user datagram protocol

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);
```

DESCRIPTION

UDP is a simple, unreliable datagram protocol used to support the **SOCK_DGRAM** socket type for the internet protocol family. UDP sockets are connectionless, and are normally used with the **sendto()** and **recvfrom()** calls (see *send*(2) and *recv*(2). The **connect()** call can also be used to simulate a connection (see *connect*(2); when used in this manner, it fixes the destination for future transmitted packets (in which case the **send()** or **write()** system calls can be used), as well as designating the source from which packets are received. The **recv()** and **read()** calls can be used at any time if the source of the message is unimportant.

UDP address formats are identical to those used by TCP. In particular, UDP provides a port identifier in addition to the normal Internet address format. Note that the UDP port domain is separate from the TCP port domain (in other words, a UDP port cannot be connected to a TCP port).

The maximum message size for a UDP datagram socket is 58 254 bytes. The default message size is 9216 bytes. The outbound and inbound message sizes for UDP sockets can be altered by using the **SO_SNDBUF** and **SO_RCVBUF** options of the **setsockopt()** system call (see *getsockopt*(2) for details).

DIAGNOSTICS

A socket operation may fail with one of the following errors returned in *errno*:

[EISCONN]	Attempt to establish a connection on a socket which already has one, or attempt to send a datagram with the destination address specified, but the socket is already connected.
[ENOBUFS]	The system ran out of memory for an internal data structure.
[EADDRINUSE]	Attempt to create a socket with a port which has already been allocated.
[EADDRNOTAVAIL]	Attempt to create a socket with a network address for which no network interface exists.

DEPENDENCIES

This entry describes the use of the UDP protocol as it applies to the Berkeley Interprocess Communication utility.

AUTHOR

udp was developed by the University of California, Berkeley.

SEE ALSO

getsockopt(2), recv(2), send(2), socket(2), inet(7F), socket(7).

NFS Services

mount(1M).

NAME

UNIX - Local communication domain protocol

SYNOPSIS

```
#include <sys/types.h>
#include <sys/un.h>
```

DESCRIPTION

The local communication domain protocol, commonly referred to in the industry as the **Unix domain protocol**, utilizes the path name address format and the **AF_UNIX** address family. This protocol can be used as an alternative to the internet protocol family (TCP/IP or UDP/IP) for communication between processes executing on the same node. It has a significant throughput advantage when compared with local IP loopback, due primarily to its much lower code execution overhead. Data is looped back at the protocol layer (OSI Level 4), rather than at the driver layer (OSI Level 2).

The HP-UX implementation of the local communication domain protocol supports all of the BSD networking system calls except for **getpeername()** and **getsockname()** (see *getpeername*(2) and *getsockname*(2)). In addition, the **MSG_OOB** flag to **recv()** is not supported (see *recv*(2)).

Addressing

AF_UNIX socket addresses are path names. They are limited to 92 bytes in length, including a terminating null byte. Calls to **bind()** to an **AF_UNIX** socket utilize an addressing structure called **struct sockaddr_un** (see *bind*(2)). Pointers to this structure should be used in all **AF_UNIX** socket system calls wherever they ask for a pointer to a **struct sockaddr**.

The include file <**sys/un.h**> defines this addressing structure.

There are two fields of interest within this structure. The first is *sun_family*, which must be set to **AF_UNIX**. The next is *sun_path*, which is the null-terminated character string specifying the path name of the file associated with the socket (for example, **/tmp/mysocket**).

Only the passive (listening) socket must bind to an address. The active socket connects to that address, but it does not need an address of its own.

For additional information on using **AF_UNIX** sockets for interprocess communication, refer to the BSD IPC programmer's manual.

Socket Buffer Size

The maximum buffer size for stream and datagram sockets is 58 254 bytes. For stream sockets, the default send and receive buffer size is 8192 bytes. The default outbound and inbound message size for datagram sockets is 9216 bytes. The send and receive buffer sizes can be altered by using the **SO_SNDBUF** and **SO_RCVBUF** options of the **setsockopt()** system call. Refer to *getsockopt*(2) for details.

AUTHOR

UNIX was developed by the University of California, Berkeley.

SEE ALSO

getsockopt(2), socket(2).

Index to Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Index
Volume 3

Printed in USA E0892

Manual Part No.
B2355-90033

Manufacturing
Part No.
B2355-90033

B2355-90033