HP-UX Reference

Vol 3: Sections 1M, 4, 5, 7 and 9

HP 9000 Series 300/800 Computers
HP-UX Release 7.0

HP Part Number 09000-90013

Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

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.

Printing History

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

> September 1989. First Edition. This manual replaces manual part number 09000-90009, and is valid for HP-UX Release 7.0 on both Series 300 and Series 800 systems.

Notes

Table of Contents
for
Volume 3

Table of Contents
Volume 3

Section 1M: Administration Commands

Entry Name(Section) *name* **Description**

Entry Name(Section) *name* **Description**

Entry Name(Section) *name* **Description**

Section 4: File Formats

Section 5: Miscellaneous

Section 7: Device (Special) Files

Section 9: Glossary

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 Administrator Manual 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*)] [*cmdarg*(*s*)]
where:

name	The name of an executable file.
option	– *noargletter*(*s*) or, – *argletter*<>*optarg* where <> is optional white space.
noargletter	A single letter representing an option without an argument.
argletter	A single letter representing an option requiring an argument.
optarg	Argument (character string) satisfying preceding *argletter*.
cmdarg	Path name (or other command argument) *not* beginning with – or, – by itself indicating the standard input.

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (see *wait*(2) and *exit*(2)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

SEE ALSO

getopt(1), getopt(3C), hier(5).

HP-UX System Administrator Manual.

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*(1) to accept requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*.

Reject prevents *lp*(1) from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*. The following option is useful with *reject:*

–r[*reason*] Associates a *reason* with preventing *lp*(1) from accepting requests. This *reason* applies to all printers mentioned up to the next **–r** option. *Reason* is reported by *lp*(1) when users direct requests to the named *destinations* and by *lpstat*(1). If the **–r** option is not present or the **–r** option is given without a *reason*, a default *reason* is used. The maximum length of the *reason* message is 80 bytes.

In the HP Clustered environment, all printers are attached to the root server and all spooling is handled as if the cluster were a single system. Remote spooling applies to spooling from or to machines outside of the cluster.

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).

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.

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. *Acctsh*(1M) describes the set of shell procedures 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 may 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. 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)). *Reason* must be a string of 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, the accounting software will collect 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 chapter in *HP-UX System Administrator Manual.*

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. The *options* are:

−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 may 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 will be total usage except number of times executed, CPU minutes, and real minutes which will be split into prime and non-prime.

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

```
acctcms file ... >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).

BUGS

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 (SEC), CPU (SEC), 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 prepended with 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.

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. The statistics will be 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 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:

(total CPU time)/(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 may 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* may 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*.

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

−E *time* Select processes ending at or before *time*. Using the same *time* for both **−S** and **−E** shows the processes that existed at *time*.

−n *pattern* Show only commands matching *pattern* that may be 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 that exceeds *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

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

BUGS

Acctcom only reports on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time, then *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. The *options* are:

–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 assuring 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*(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 *acctprc*(1M) commands:

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

FILES

/etc/wtmp

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).

BUGS

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

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* are:

- **−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
```

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).

EXTERNAL INFLUENCES

Environment Variables

For the output of *acctprc2*, if the user ID's 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)).

BUGS

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*(1M), for example. More precise conversion can be done by faking login sessions on the console via the *acctwtmp* program in *acct*(1M).

CAVEAT

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* will be invoked with argument *switch*. If the number of free disk blocks in the **/usr** file system falls below 500, *ckpacct* will automatically turn 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 will do a slower version of disk accounting by login directory. *Files* specify the one or more filesystem names where disk accounting will be done. If *files* are 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 (01–12). 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 insures 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 may 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

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, rfa_netunam**)
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

DEPENDENCIES

Series 800

The *rfa_netunam* system call runs on Series 800 only

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

The command analyzes and displays the audit information contained in the specified audit file(s), *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) which consists 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 the *start_time* to the end of the file.

The *audisp* command 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 the *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 diskless 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 as part of the *init*(1M) 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 superusers.

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

-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. Without arguments, *audusr* displays the command usage. *Audusr* is restricted to superusers.

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*(1M) to specify events to be audited.

–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. Furthermore, 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

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 may edit **/etc/backup** to customize it for your system. For example, *backup* uses *tcio*(1) with *cpio* to backup your files on an HP Command Set 80 disc'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 left running overnight and the tape runs out, *backup* terminates, leaving the *find* process still waiting. You should 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 [**–i**] [**–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

–i Report the number of used and free inodes.

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

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

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

NAME

boot – bootstrap process

DESCRIPTION

The bootstrap process on the Series 800 involves the execution of three software components: *pdc(1M), isl(1M),* and *hpuxboot(1M).* After the processor is **RESET,** *pdc,* the processor dependent code or 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 *hpuxboot* utility, the HP-UX specific bootstrap loader. Lastly, *hpuxboot* downloads the HP-UX kernel object file from an HP-UX file system and then transfers control to the loaded kernel image.

SEE ALSO

hpuxboot(1M), isl(1M), pdc(1M).

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*(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.

The *brc* procedure clears the mounted file system table, **/etc/mnttab** (see *mnttab*(4)), and loads any programmable micro-processors with their appropriate scripts.

The *bcheckrc* procedure performs all the necessary consistency checks to prepare the system to change into multi-user mode. It will prompt to set the system date and to check the file systems with *fsck*(1M).

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

The *rc* procedure 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.

The *powerfail* procedure is invoked when the system detects a power failure condition. Its chief duty is to reload any programmable micro-processors with their appropriate scripts, if suitable. 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

DESCRIPTION

Without the **–d** option, the *buildlang* utility automatically sets up the language environment as specified by the *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*(3C), which make up a language definition. 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**. And 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*(3C) sets up the default "C" locale for that category (see *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 upper case characters, while the keywords are composed of lower case 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*.

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.

ui 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).

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*()).

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 "0123456789b+-.,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_12**

abmon_1 to **abmon_12**

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 have 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 the ending date of the era extends to the beginning of time while **+*** indicates it extends to the end of time. The ending date may chronologically be 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 fo 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 can be used
            <0113 0153>   <0114 0154>       # octal constants can be used
            < 77  109 >   < 78  110 >       # decimal constants can be used
            < 'O' 'o' >   < 'P' 'p' >       # literal constants can be used
            < '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
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

−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 only supports 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.

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**] [**–n**] [**–w**] [**–z**] [sections]

DESCRIPTION

Catman creates the formatted versions of the online manual from the *nroff*(1) 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 **cat*.Z** directory. If any changes are made, *catman* will recreate the **/usr/lib/whatis** database.

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

If there are parameters not starting with a "**–**", they are taken to be a list of manual sections to look in. For example:

catman 123

will cause the updating to happen only to manual sections 1, 2, and 3.

Options

–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.

AUTHOR

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

FILES

/usr/man/man*[.Z]/*	raw (*nroff*(1) source) manual pages [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).

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 given 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*.

Notice that:

```
chroot newroot command >x
```

will create the 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 the super-user.

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.

WARNINGS

Command cannot be in a shell script.
One should 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.

SEE ALSO

chdir(2), chroot(2).

EXTERNAL INFLUENCES

International Code Set Support

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

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. After *clri* is executed, any blocks in the affected file will show up as "missing" in an *fsck*(1M) of the *file-system*. 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.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to zero out an inode which does appear in a directory, care should be taken to track down 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 will destroy the new file. The new entry will again point to an unallocated inode, so the whole cycle is likely to be repeated again and again.

SEE ALSO

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

BUGS

If the file is open, *clri* is likely to be ineffective.

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 *line* specified. The *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

The *cluster* command allocates kernel resources and notifies the kernel that operation of an HP Cluster is intended. The *cluster* command can only be executed by the machine listed as the root server in the **/etc/clusterconf** file. The machine executing the *cluster* command is converted from a stand-alone system to the root 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 root 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** *file*] [**-l** *file*] *dfile*
/etc/config -a

DESCRIPTION

Config enables the user to configure the following parts of the operating system:

device switch drivers and I/O cards

root and swap devices

selected system parameters

kernel code that handles messages, semaphores, and shared memory

Config reads a user-provided description of an HP-UX system (*dfile*) and generates two output files. The first output file is a C program that defines the configuration tables for the various devices on the system. The second output file is a makefile script (see *make*(1)) that will compile the C program produced and relink the newly configured system.

The following options are available:

-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**.

This file is supplied with the HP-UX system and should not be modified unless the user fully understands its construction.

-c *file* Specify the name of the configuration table file produced by running the user-data file *dfile* through *config*(1M). The default file name is **conf.c**.

-l *file* Specify the name of the makefile script (see *make*(1)) that will compile the configuration program and relink the newly configured system. The default file name is **config.mk**.

–a Produce a script of *mkdev*(1M) templates.

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

The **-a** option and *dfile* are mutually exclusive and exactly one of these must be specified.

The first part of *dfile* allows you to configure:

device switch drivers

I/O cards

pseudo-drivers, such as *pty*

Each line has the following format:

devname

where *devname* is the driver name for the device as it appears in the alias table in **/etc/master**. For example, **cs80** selects the HP7912 64MB disc drive, **98625** selects the high speed disk interface card, and **ethernet** selects the link-level access protocol. The complete list of configurable

devices, cards, and pseudo-drivers is given in the file, **/etc/master**.

The following devices are not configurable and should not be specified in *dfile*:

swap	**cons**	**tty**
sy	**mm**	**iomap**
graphics	**r8042**	**hil**
nimitz	**ite200**	

The second part of *dfile* is optional, and can contain four different types of lines:

1. Root device specification

Each of these lines has the following format:

root *devname address*

devname The driver name for the device, as it appears in the alias table in **/etc/master**, such as cs80 for the HP7933 404MB disc drive.

address The minor device number (in hexadecimal without the preceeding **0x**).

2. Swap device specification

These are used if you want the system to auto-configure the swap device given only the swap size:

swapsize *nblocks*

where *nblocks* is the desired swap size in blocks.

If you want to specify both the swap device location and its size, the specification line takes the following form:

swap *devname address swplo* [*nswap*]

devname The driver name for the device as it appears in the alias table in **/etc/master** (for example, **cs80** for the HP7933 404MB disc drive).

address The minor device number (in hexadecimal without the preceeding **0x**).

swplo The location (in decimal) of the swap area.

A negative value (typically **-1**) for *swplo* specifies that a file system is expected on the device. At boot-up, the super block will be read to determine the exact size of the file system, and this value will be 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 that at least that much area must be reserved. Zero means to reserve no area at the head of the device. The case for *swplo* pointing beyond the end of the device is handled.

nswap The number of disc blocks (in decimal) 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 unless the user fully understands the ramifications of doing so. (See the *HP-UX System Administrator Manual* for details on each parameter.)

Each line contains two fields. Field one can have 20 characters maximum and field two 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**, **shmem** are specified as **0**, the kernel code for these features will not be included. If they are specified as **1**, the kernel code will be included; this is the default. The features may be specified independently of each other. If the code is included, the parameters listed below may 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
conf.c	Default output configuration table
config.mk	Default output *make*(1) script
mkdev	Default output *mkdev*(1M) script

SEE ALSO

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

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*(1M) on each file system.

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 may 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 – 0755
owner – bin
group – bin

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

An optional argument of **–o** will force *cpset* to move *object* to **OLD***object* in the destination directory before installing the new object.

For example:

cpset echo /bin 0755 bin bin

cpset echo /bin

cpset echo /bin/echo

All the examples above have the same effect (assuming the user is an administrator). The file **echo** will be copied into **/bin** and will be given **0755, bin, bin** as the mode, owner, and group, respectively.

Cpset utilizes the 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 will be responsible for defining the "official" locations of the source.

Cross Generation

The environment variable **ROOT** will be 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*(1) to set optional ACL entries on a newly installed file.

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 *non-existent 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 may 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 abiguous 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.

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).

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).

STANDARDS CONFORMANCE

cron: SVID2

NAME

csp – create cluster server processes

SYNOPSIS

/etc/csp [[**–a** | **–d**] *number*]

DESCRIPTION

The *csp* command changes the number of HP Cluster server processes (CSPs) running on the system. When no arguments are given, *csp* reads the **/etc/clusterconf** file to determine how many CSPs should be running.

The **–a** option adds the specified *number* of CSPs to the number currently running.

The **–d** option deletes the specified *number* of CSPs from the number currently running.

If neither the **–a** nor **–d** options are given, the number of CSPs running is set to *number*.

The *csp* command creates the minimum *number* of CSPs to run on a cnode. Additional CSPs can be started automatically when system load demands it. The *csp* command is normally executed from the **/etc/rc** script.

This command can be executed only by the superuser.

AUTHOR

Csp was developed by HP.

FILES

/etc/clusterconf

SEE ALSO

clusterconf(4), cluster(1M).

NAME

decode – read and decode diagnostic events from the error log

SYNOPSIS

decode [**–L** *logfile*] [**–d** *devicefile*] [**–e** *phys_path*] [**–m** *major*] [**–n** *driver*] [**–p** *port*] [**–t** *n*] [**–w**]

DESCRIPTION

Decode reads a series of encoded diagnostic event messages, selects some of them based on the options, and writes them in a human-readable format to the standard output.

By default, if the standard input is a terminal, the event messages are read from the default log file **/usr/adm/diaglog**. If the standard input is not a terminal, the event messages are read from the standard input.

By default, all the logged events are shown.

Options

-L *logfile* — Read events from the named *logfile* instead of from the standard input or the default log file.

–d *devicefile* — Restrict the output to describe only events pertaining to the named device file.

–e *phys_path* — Restrict the output to describe only events pertaining to the specified physical hardware path. The *phys_path* is expressed in the **1.2.3** notation.

–m *major* — Restrict the output to describe only events pertaining to devices with the specified major number.

–n *driver* — Restrict the output to describe only events pertaining to devices using the named driver.

–p *port* — Restrict the output to describe only events occuring on the specified port number.

–t *n* — If *n* is a positive integer, *decode* shows the last *n* selected events in the *logfile*.

If *n* is a negative integer, *decode* skips the first *n* selected events from the beginning of the file, and then shows any remaining selected events. If there are fewer than *n* selected events in the file, nothing is shown.

–w — Wait for a newline to be entered from the standard input after displaying each event, so that the user can look at it before continuing. If the user enters a line beginning with 'q' (quit), *decode* exits immediately. This option cannot be used if the diagnostic events are being read from the standard input.

Each event written to standard output is in the following format:

Diagnostic event — A number identifying the entry of the diagnostic event, assigned sequentially from the beginning of the log file.

Time of event — The date and time that *diag0*(7) processed the event.

Reporting entity — Manager name of reporting port number, derived from the I/O tree.

Port number — Port number of the I/O subsystem entity that reported the event.

Physical path — Physical path derived from the I/O tree and the manager-specific information.

Classification — Hardware event or software event.

Low-level I/O status The unique number used to identify the status error and the I/O subsystem entity detecting the error.

Hardware status Numerical data expressed as bytes in a hexadecimal string.

Manager-specific information
Numerical data expressed as bytes in a hexadecimal string.

Other fields might appear, depending on the reporting entity. Specific manager information and hardware status might also vary.

RETURN VALUE

If the named *logfile* cannot be opened, *decode* exits with a non-zero status.

WARNINGS

Decode attempts to decode the device manager-specific information for some diagnostic events into human-readable form. In other cases, it displays the information in hexadecimal format.

AUTHOR

Decode was developed by HP.

FILES

/dev/diag0
/usr/adm/diaglog

SEE ALSO

delog(1M), diag0(7).

NAME

delog – diagnostic event logger for I/O subsystem.

SYNOPSIS

delog [[**-e** [number]] | [**-w**]] [**-b**] [**-d** dev file]
[**-f** outfile] [**-L** error log] [**-U** vmunix] [**-V** version]

DESCRIPTION

In general, *delog* only reads in an event message from the diagnostic driver and then writes it to a log file, with no information being sent to the system console. The event message can then by decoded into human-readable format at some later time by *decode*. By using various options *delog* can write a synopsis of the messages to the system console and/or to a designated file. These options, as described below, allow the user to tune the output of *delog* to the needs of the system.

Delog accepts the following list of parameters:

-e *number*	print out a brief synopsis for only fatal diagnostic events. See below for a description of *number*. *Number* can be **1** or **2**.
-w	print out a brief synopsis of the diagnostic event.
-b	send a synopsis to both the system console and a user defined file.
-d *dev file*	set the device file path name (defaults to */dev/diag0*).
-f *outfile*	set the output file path name (defaults to */dev/null*).
-L *error log*	set the error log path name (defaults to */usr/adm/diaglog*).
-U *vmunix*	set the *vmunix* path name (defaults to */hp-ux*).
-V *version*	set the version number of the *vmunix* file (default is to be determined).

Messages sent either to the system console or a user defined file take on the following format:

Diagnostic **class type** from **XX** (at **a.b.c**) status = **x**
Hw stat = (hardware status printout as a HEX string)

Where:

class	is replaced with either hardware or software.
type	is replaced with either error or warning.
XX	is the name of the driver that reported the event.
a.b.c	is the hardware path derived from the I/O tree.
x	is an ASCII string corresponding to the LLIO status.

The **-w** option prints out, on the system console, warning and error information. Warnings are informational and are not usually significant. Errors may be informational or they may be reports of fatal errors. Therefore, a large amount of data, mostly informational, will go to the system console.

The **-e** option causes *delog* to print messages that are only I/O subsystem errors. However, because of the nature of I/O subsystem messages, not all superfluous informational messages can be suppressed. To further reduce the amount of data written to the system console, the **-e** option has a numeric parameter, *number*. The default value of *number* is **1** if nothing is specified. When *number* is **2**, the hardware status is suppressed. The format of the output for the **-e** option is the same as that of the **-w** option. (Note: the **-e** and **-w** options are mutually

exclusive.)

The synopsis message can be redirected to a file using the **-f** option. In combination with the **-b** option the messages can be sent to the system console and the designated file. If the **-b** option is used by itself, the result is the same as if the **-w** option were specified.

Because the read function of the diagnostic driver is exclusive, only one *delog* logging process can be run at a time.

RETURNS

If the diagnostic driver cannot be opened, *delog* will exit with a non-zero status. If the log file cannot be opened, *delog* will not complain, instead *delog* prints a synopsis of the event on the system console (and the user defined file, if specified) and continues.

WARNINGS

Delog does not do any error log manipulation. If the file used for error logging becomes too large, it is up to a system administrator to clear it out.

AUTHOR

Delog was developed by HP.

FILES

/etc/rc
/dev/diag0
/usr/adm/diaglog
/hp-ux

SEE ALSO

decode(1M), diag0(7).

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.

This command is most commonly used by **/etc/rc** (see *brc*(1M)) to construct a mount table entry for the root device.

EXAMPLES

The command:

/etc/devnm /usr

produces

/dev/dsk/c1d0s9 /usr

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

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

brc(1M), setmnt(1M).

STANDARDS CONFORMANCE

devnm: SVID2

NAME

df − report number of free disk blocks

SYNOPSIS

df [**−t**] [**−f**] [**−b**] [file-systems]

DESCRIPTION

Df prints out the number of free 512-byte blocks and free inodes available for online file systems by examining the counts kept in the super-block(s). *File-systems* may be specified either by device name (e.g., **/dev/dsk/0s1**) or by mounted directory name (e.g., **/usr**). If the *file-systems* argument is unspecified, the free space on all of the mounted file systems is printed.

The **−t** flag causes the total allocated block figures to be reported as well.

If the **−f** flag is given, only an actual count of the blocks in the free list is made (free inodes are not reported). With this option, *df* will report on raw devices.

If the **−b** flag is given, the total number of blocks available for swapping as well as the number of blocks free for swapping are reported.

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 super-user. See *fs*(4) for information about *fs_minfree*.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

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

STANDARDS CONFORMANCE

df: SVID2, XPG2, XPG3

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

–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 returns 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 does not support Amigo disks.

SCSI

The SCSI specification provides 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's operator manual accompanying the unit.

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.

p Tables suitable for inclusion in the device driver are output.

d Tables suitable for generating the disk description file **/etc/disktab** are output.

b *block size* Defines a block size in bytes to be used as the sector size in generating the above tables. Legal values for *blocksize* are **256**, **512**, **1024** and **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* will prompt 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.

The 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 MB, *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 out the requested tables. All information printed out 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 wishes to see the device driver tables output for the 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 the hp7945 is an unknown disk name, *disksecn* will prompt 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.

RETURNS

Disksecn returns **0** for a successful completion, **1** for a usage error, **2** for a user not wanted to input parameters for an unknown disk and **3** for a disk that is too small or an invalid block size.

Disksecn will abort and print out an error message under the following conditions:

Disksecn is invoked without specifying a disk name.

Both a **-p** and a **-d** switch were requested.

A block size other than those specified as legal is requested.

An unknown disk name is specified and user does not wish to supply disk information.

A disk whose maximum storage space is less than approximately 44 MB.

WARNINGS

When using the **-d** switch alternate names are not included in the output

There must be spaces typed between each of the switches in the command line when invoking *disksecn*.

There must be a space between the **-n** switch and the disk name argument to that switch. For example:

disksecn -p -b 1024 -n hp9712

At this point in time the program has no way to save the block size that was 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* output lines on the standard output, one per user, in the following format:

uid login #blocks

where

uid - the numerical user ID of the user;

login - the login name of the user; and

#blocks - the 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.

Diskusg recognizes the following options:

-s the 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 (see *labelit* on *volcopy*(1)).

-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 will generate 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 – flag is given, then *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*(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. Because *dmesg* is run only every few minutes, not all error messages are guaranteed to be logged.

AUTHOR

Dmesg was developed by the University of California, Berkeley, California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/adm/messages	error log (conventional location)
/usr/adm/msgbuf	scratch file for memory of – option

NAME

dump, rdump – incremental file system dump, local across network

SYNOPSIS

/etc/dump [*key* **[** *argument ...* **]** *filesystem* **]**
/etc/rdump [*key* **[** *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**. The *key* specifies the date and other options about the dump. *Key* consists of characters from the set **0123456789dfnsuWw**.

0–9 This number is the "dump level". All files modified since the last date stored in the 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.

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. In case of *rdump* this key 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 the group **operator** by means similar to *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* will wait for reels to be changed. The default tape size is 2300 feet.

u If the dump completes successfully, write the date of the beginning of the dump on file **/etc/dumpdates**. 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 *ctime*(3C) format dump date. The file **/etc/dumpdates** may be edited to change any of the fields, if necessary.

W Print out the most recent dump date and level for each file system in **/etc/dumpdates**, 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, the *key* 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 these conditions: end of tape, end of dump, tape-write error, tape-open error or disk-read error (if there are more than a threshold of 32). In addition to alerting all operators implied by the **n** key, *dump* and *rdump* interact with the control terminal operator when it can no longer proceed or if something is grossly wrong, by posing questions requiring "yes" or "no" answers.

Since making a full dump involves considerable time and effort, *dump* and *rdump* checkpoint itself at the start of each tape volume. If for any reason writing that volume fails, *dump* and *rdump* will, with operator permission, restart itself from the checkpoint after the old tape has been rewound and removed, and a new tape has been mounted.

Dump and *rdump* report information to the operator periodically, 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* are busy, and will be for some time.

EXAMPLES

In the following example, assume that the file system */mnt* is to be attached to the file tree at 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 6250BPI.

```
/etc/dump 0df 6250 /dev/rmt/0h /mnt
```

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 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 6250BPI.

```
/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	to find group **operator**

SEE ALSO

restore(1M), rmt(1M), acl(5), checklist(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 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*(1) by including an entry for *ecclogger* in the *crontab*(1M) file for root. The recommended frequency for running *ecclogger* is once per hour. (*Eccscrub* is automatically invoked by *ecclogger* whenever an error is corrected.)

Eccscrub is a utility that 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*(1) once per day.

In order to achieve the recommended frequencies, the following two entries need to be added to the *crontab*(1M) 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

fbackup – selectively backup files

SYNOPSIS

/etc/fbackup **–f** *device* [**–f** *device*] ... [**–0-9**] [**–uvyAH**] [**–i** *path*] [**–e** *path*] [**–g** *graph*] [**–I** *path*] [**–c** *config*]

/etc/fbackup **–f** *device* [**–f** *device*] ... [**–R** *restart*] [**–uvyAH**] [**–I** *path*] [**–c** *config*]

DESCRIPTION

Fbackup combines features of *dump*(1M) and *ftio*(1), to provide a flexible, high-speed file system backup mechanism. *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. Generally, the output device is a raw magnetic tape drive, but it can also be the standard output or a file. By explicitly specifying trees of files to be included or excluded from an *fbackup* session, the user can construct an arbitrary graph of files. *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.

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 *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**. 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 *nl_ctime*(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.

Each volume contains an index consisting of a list of all files in the graph being backed up. Each file entry contains the volume number and the path name of the file. At the beginning of every volume, *fbackup* assumes that all files which have not already been backed up will fit on that volume, which is an erroneous assumption for all but the last volume. Hence, the index on the first volume reports that all files are on volume one, and the index on the last volume is the only index which is completely accurate (see WARNINGS section). Indices that are not on the last volume are accurate for all previous volumes of the same set.

When using 9-track tape drives, *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.

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.

Options

–c *config* *Config* is interpreted as the name of the configuration file. This file can contain values for the following parameters:

- Number of 512 byte blocks per record,
- Number of these records of shared memory to allocate,

- Number of records between checkpoints,
- Number of file reader processes,
- Maximum number of times *fbackup* will 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 may 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.

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 32, 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 may be used is set to 100, the file to be executed at volume change time is **/usr/adm/fbackupfiles/chgvol**, and the file to be executed when a fatal error occurs is **/usr/adm/fbackupfiles/error**.

blocksperrecord	32
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**

Each value listed is also the default value, except **chgvol** and **error**, which default to null values.

−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 the number of times that 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. 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. 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.

−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 a path name of a tree. Lines not beginning with **i** or **e** are ignored. There is no default graph file. For example, if a user wants 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 the number of times that the **−i** option can be specified.

−u Update **/usr/adm/fbackupfiles/dates** 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), and *fbackup* must complete successfully.

−v Run in verbose mode. Status messages that are not normally seen are generated.

−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*(2)) 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 online 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.

−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, **/usr/adm/fbackupfiles/dates** 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, you must use the **−A** option to enable its recovery on a system lacking access control list functionality.

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 wishes to maintain a database of past *fbackup* sessions so that incremental backups are possible.

If sufficient online 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 index files online 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.

WARNINGS

Because of present file system limitations, files whose inode data, but not their contents, are modified while a backup is in progress might not be included in 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.

Indices are completely 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 online index, which is produced after the last volume has been written.

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 might be consumed.

The user of *fbackup* need not be the super-user. However, if the user does not have access to a given file, the file is not backed up.

Fbackup consists of multiple executable objects, and it expects all of these to reside in /etc.

Graph file names are compared character by character when checking the **/usr/adm/fbackupfiles/dates** 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.

DEPENDENCIES

RFA and 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

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).

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, *adjust* 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.

NAME

frecover – selectively recover files

SYNOPSIS

/etc/frecover **–r** [**–hosvyAFOX**] [**–c** *config*] [**–f** *device*] [**–S** *skip*]

/etc/frecover **–R** *path* [**–f** *device*]

/etc/frecover **–x** [**–hosvyAFOX**] [**–c** *config*] [**–e** *path*] [**–f** *device*] [**–g** *graph*] [**–i** *path*] [**–S** *skip*]

/etc/frecover **–I** *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**, 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/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, 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*.

–R *path* An interrupted 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.*

The following characters may 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 the number of times that the **–e** option may be specified.

−f *device* *Device* identifies the backup device instead of **/dev/rmt/0m**, which is the default. If *device* is "-", *frecover* reads from standard input. Thus *fbackup*(1M) and *frecover* can be used in a pipeline backup and recover a file system with the command:

fbackup -i /usr -f - | (cd /mnt; frecover -Xrf -)

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.

−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

−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 the number of times that the **−i** option may be specified.

−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 the 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.

−O Do not attempt to change the ownership of the recovered files.

−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(s) 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.

WARNINGS

The user of *frecover* need not be the superuser. However, if a user does not have access to a given file, the file will not be recovered.

DEPENDENCIES

Series 800

Access control lists and the **−A** option are not implemented.

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).

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).

NAME

fsck – file system consistency check and interactive repair

SYNOPSIS

/etc/fsck **–p** [*file system ...*]
/etc/fsck **–P** [*file system ...*]
/etc/fsck [**–b** *block#*] [**–y**][**–n**][**–q**][*file system ...*]

DESCRIPTION

Fsck audits and interactively repairs inconsistent conditions for HP-UX file systems. 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 used.

Fsck checks a default set of file systems or the file systems specified in the command line. If *file system* 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 **-p** option is used without specifying a *file system*, *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 will be checked on pass 1, and other "root" ("0" section) file systems on pass 2. Other small file systems are checked on separate passes (e.g. 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 (e.g. 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* will preen 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 will correct; if it encounters other inconsistencies it exits with an abnormal return status. For each corrected inconsistency, one or more lines will be printed identifying the file system on which the correction will take place, and the nature of the correction. The inconsistencies that are corrected 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 those file systems which were cleanly unmounted will not be checked (see *fsclean*(1M)). This can greatly decrease the amount of time required to reboot a system which 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 will result in a loss of data. The amount and severity of data lost may 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* will default to a **–n** action. The following options are interpreted by *fsck*.

−b Use the block specified immediately after the flag 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. (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 as this is a free license to continue after essentially unlimited trouble has been encountered.

−n Assume a **no** response to all questions asked by *fsck;* do not open the file system for writing.

−q Quiet *fsck*. Do not print size-check messages in Phase 1. Unreferenced fifos will silently be removed. If *fsck* requires it, counts in the superblock and cylinder groups will be automatically fixed.

In all cases the inconsistencies checked by *fsck* are as follows:

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:
   Directory size not of proper format.
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 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.
11. 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 preexist 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, and 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), since access control list information may have been lost and should be corrected.

After *fsck* has checked and fixed the file system, it will store the correct fs_clean flag in the super block if it is not already there. For a non-root file system, FS_CLEAN will be 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 there were no problems found and if FS_OK was already set.

Checking the raw device is almost always faster.

RETURN VALUE

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.

DEPENDENCIES

Series 300

There is only one section per volume.

Series 800

Continuation inodes and access control lists are not implemented.

AUTHOR

Fsck was developed by HP, AT&T, and the University of California, Berkeley.

FILES

/etc/checklist contains default list of file systems to check.

SEE ALSO

fsclean(1M), mkfs(1M), newfs(1M), checklist(4), fs(4), inode(4), acl(5).

NAME

fsclean – determine shutdown status of specified file system

SYNOPSIS

/etc/fsclean [-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 of *checklist* must be present for *fsclean* to be able to check each file system.

Fsclean reads the super block to determine if the file system's last shutdown was done correctly. If all of the checked file systems were shutdown correctly, *fsclean* returns **0**. If any were not, *fsclean* returns **1**. Any other errors, such as "cannot open the specified device file," return **2**. *Fsclean* is normally silent.

The only option is:

–v Be verbose. Prints the fs_clean value of each file system checked.

AUTHOR

Fsclean was developed by HP.

FILES

/etc/checklist

SEE ALSO

checklist(4), brc(1M), reboot(1M).

NAME

fsdb – file system debugger

SYNOPSIS

/etc/fsdb special [**–b block#**] [–]

REMARKS

Always execute *fsck*(1M) after having used *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 will be 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 fragment 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 may be used as function delimiters but are not necessary. A line with just a new-line character will increment 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:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size in byte unit
a#	data block numbers (0 – 14)
at	time last accessed
mt	time last modified
ct	last time inode changed
maj	major device number
min	minor device number
ci	continuation inode 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 5th 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

The 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.

DEPENDENCIES

Series 300 Diskless

The following mnemonic is also used:

cno	cnode ID for a device file

Series 800

Continuation inodes are not implemented.

AUTHOR

Fsdb was developed by HP and AT&T.

SEE ALSO

fsck(1M), stat(2), dir(4), fs(4).

NAME

fstomnt – convert file system checklist file to new format

SYNOPSIS

/etc/fstomnt [**–r**] [*file*]

DESCRIPTION

Fstomnt converts entries in old *checklist*(4) format to entries in new *checklist*(4) format. Input is read from the **/etc/checklist** file unless the optional *file* argument is given. The converted entries are written to the standard output.

The **–r** option reverses the direction of the conversion. That is, entries in new *checklist*(4) format are converted to old *checklist*(4) format.

The old *checklist*(4) format contains a list of mountable file system entries. The fields within each entry are separated by one or more blanks. Each file system entry is contained on a separate line. An entry in old *checklist*(4) format is defined by the following fields:

special file name
block special file name
directory
type
pass number
backup frequency
comment

special file name is either a character or block special file name. The *special file name* field is required.

The remaining fields are optional. However, if any optional fields are specified, all remaining fields are required.

block special file name
is the block special file corresponding to *special file name* .

directory is the name of the root of the mounted file system that corresponds to the *block special file name*.

type can be **rw**, **ro**, **sw** or **xx**. When the *type* field is converted to new format, **rw** and **ro** are unchanged, **sw** becomes **swap**, and **xx** becomes **ignore**.

pass number is used by the *fsck*(1M) command to determine the order in which file system checks are done when using the **-p** option of *fsck*(1M).

backup frequency
is reserved for possible use by future backup utilities.

comment is an optional field that starts with a pound sign (#) and ends with a newline.

The new *checklist*(4) format is defined on the *checklist*(4) page.

DIAGNOSTICS

Illegal *checklist*(4) entries are written as comment lines to the standard output.

EXAMPLES

An old format entry of the form:

```
/dev/rdsk/0s0   /dev/dsk/0s0   /        rw 1 0   #root disk
/dev/rdsk/0s1   /dev/dsk/0s1   /spare   xx 1 0   #ignored entry
```

will be converted to a new format entry of the form:

```
/dev/dsk/0s0   /        hfs      rw         0 1   #root
/dev/dsk/0s1   /spare   ignore   defaults   0 1   #ignored entry
```

WARNINGS

When converting from new to old format, entries of non-**hfs** type will be ignored. Also, only the **rw** and **ro** options are significant. All other options are ignored.

Beware of

/etc/fstomnt > /etc/checklist

as this will destroy the **checklist** file before reading it.

AUTHOR

Fstomnt was developed by HP.

FILES

/etc/checklist

SEE ALSO

checklist(4), fsck(1M).

NAME

fuser, cfuser − identify processes using a file or file structure

SYNOPSIS

/etc/fuser [−**ku**] files [−] [[−**ku**] files]

/etc/cfuser [−**ku**] files [−] [[−**ku**] files]

DESCRIPTION

Fuser lists the process IDs of the processes using the *files* specified as arguments. 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. PP 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 the super-user can terminate another user's process (see *kill*(2)). Options may 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 seperate 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 the super-user on a Series 300 system, will terminate all processes that are preventing disk drive one from being unmounted, listing the process ID and login name of each process being killed.

fuser −u /etc/passwd
: will list process IDs and login names of processes that have the password file open.

fuser −ku /dev/dsk/1s? −u /etc/passwd
: if typed by the super-user on a Series 800 system, 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 namelist

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. In this way, *wtmpfix* and *acctcon1* will not factor in a date change record pair more than once.

In addition to correcting time/date stamps, *wtmpfix* will check 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 will change the login name to **INVALID** and write a diagnostic to the standard error. In this way, *wtmpfix* reduces the chance that *acctcon1* will fail when processing connect accounting records.

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).

DIAGNOSTICS

Wtmpfix generates these diagnostics:

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 (not seen yet)

BUGS

Fwtmp generates no errors, even on garbage input.

STANDARDS CONFORMANCE

fwtmp: SVID2

wtmpfix: SVID2

NAME

getty – set terminal type, modes, speed, and line discipline

SYNOPSIS

/etc/getty [**–h**] [**–t** timeout] line [speed [type [linedisc]]]
/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.

Line is the 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* will force 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* will not force a hangup on the line since the driver ignores changes to zero speed on ports open in direct mode (see *modem*(7)). In addition, when *getty* is invoked with the **–h** flag, *getty* will not force a hangup on the line before setting the speed to the default or specified speed.

The **–t** flag plus *timeout* in seconds, specifies that *getty* should exit if the open on the line succeeds and no one types anything in the specified number of seconds. The optional second argument, *speed*, is 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. The optional third argument, *type*, is 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. The optional fourth argument, *linedisc*, is a character string describing which line discipline to use in communicating with the terminal. Again the 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 will cause *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 may be typed after the login name. These are passed to *login*, which will place them in the environment (see *login*(1)).

A check option is provided. When *getty* is invoked with the **−c** option and *file*, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *ioctl*(2) to interpret the values. Note that some values are added to the flags automatically.

DEPENDENCIES

HP 2334 MultiMux:

The modem control parameter *MRTS* must be present in the **/etc/gettydefs** file when using *getty* in conjunction with an HP 2334 or HP 2335 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 HP 2334 or HP 2335 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 login via *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 very similar to *getty*(1M) in function, 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. HP 2334A 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

The log file used is **/usr/spool/uucp/X25LOG**.

AUTHOR

Getx25 was developed by HP.

SEE ALSO

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

UUCP, a tutorial in *HP-UX Concepts and Tutorials.*

NAME

hpux – HP-UX bootstrap and installation utility

SYNOPSIS

hpux [**–a** devicefile]... [**–f**number] [**–i**string] [**–o**] [**–r**]
[**–m**[**p**|**s**]] [**boot**] [devicefile]
hpux install
hpux ls [devicefile]
hpux –v
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 three operations: **boot**, **install**, and **ls**. The last operation listed, **copy**, is obsolete and may not work on your system. The **boot** operation loads an object file from an HP–UX file system or raw device and transfers control to the loaded image. The **install** operation is used during first time installation. It attempts to execute an install image from a properly formatted installation device. The **ls** operation lists the contents of HP–UX directories in a format similar to *ls*(1). The **copy** operation copies data between HP–UX files and/or raw devices. This operation is not recommended since it may damage your file systems.

The command sequences for all operations are presented under **SYNOPSIS** above. They may be either entered interactively to *isl* or present in an *isl autoexecute* file. The first sequence performs the **boot** operation. The second performs the **install** operation. The third performs the **ls** operation. The fourth returns the release and version numbers of *hpux*. The last performs the **copy** operation.

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**, **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), for example **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 **tape1** 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. In a *devicefile* specification, the only required component is the hardware path. All other components are optional. 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**, **disc2**, **disc1**, **lan1**, and **tape1** managers. **Disc0** manages all

CS–80 discs connected via HP–IB, including cartridge tape devices, and **disc2** manages all CS–80 discs connected via the HP 27111. **Disc1** manages all CS–80 discs connected via NIO HP–IB, including cartridge tape devices. **Lan1** manages remote boot through the HP28652A NIO based lan card. Remote boot is currently supported on this card only and not on any CIO based lan card. **Tape1** manages the HP 7974, HP 7978, and HP 7980 tape drives.

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 HP 27111 bus address.

Minor Numbers

The minor number, *n*, in a *devicefile* specification controls driver dependent functionality. The HP-UX System Administrator Manual describes the specific minor number encodings for the 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**, **disc2**, **disc1**, **lan1**, and **tape1** refer to the HP-UX System Administrator Manual.

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** 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**, and **disc2**, they are relative to the section specified in the minor number (in the device part) of the *devicefile* specification.

BOOT

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.

The default *devicefile* for **boot** has a minor number and skip count of zero, and a file name of hp–ux. The hardware path and manager are derived from the Primary Boot Path maintained by *pdc*(1M) or interactively given to *pdc*. For example, if *pdc* loaded *isl* from a CIO channel at MID–BUS module 8, the HP-IB card at CIO slot 0, and the CS–80 disc at address 0, the default *devicefile* would be disc0(8.0.0;0)hp–ux. For more details, consult the appropriate HP 9000 Series 800 architecture document.

Any missing components in a specified *devicefile* are supplied from the default above. 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 disc 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** *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. Note that the use of **−a** 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.

−r This option is not currently supported. It was previously used as a debugging tool.

−m[p|s] 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.

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

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 the same as the one specified to *pdc*.

COPY

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

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*. *Pdc* will then

ask if the primary boot path is acceptable. Answering yes (**Y**) is usually appropriate. *Pdc* will then load *isl* and *isl* will interactively prompt 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* may be used from *isl* interactively, they may also be executed from an *autoexecute* file. Refer to the appropriate documentation for building *autoexecute* files. In the examples below, all user input is in boldface type.

Default Boot

ISL> **hpux**

Boot : disc0(8.0.0;0x0)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 is **0** specifying section 0 of the disc, 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 disc. 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) disc. 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 HP 7914 disc 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 Disc

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 is specified. All other values are boot defaults. The

object file comes from the file system in section 0 of the disc, 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 diskless 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 -a tape1(8.2.0)**

Boot : disc0(8.0.0;0x0)hp-ux : 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. The only other devices configured are the console and root device, which **boot** derived from *pdc* .

First–Time Installation

ISL> **hpux install**

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 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) will default to installing from cartridge tape while larger systems (for example, model 850) default to reel tape.

Listing Directory Contents

ISL> **hpux ls**

```
Ls : disc0(8.0.0;0x0).
d rw x r— x r— x      9      2048 ./
d rw x r— x r— x      6      2048 ../
d rw x r— x r— x      2      4096 lost+found/
—rw—r w —r——          1       746 .profile
d rw x r w x r— x      2      1024 bin/
d rw x r— x r— x     12      1024 dev/
d rw x r w x r— x      5      1024 etc/
d rw x r w x r w x      2        64 tmp/
d rw x r w x r— x      3      1024 usr/
—rw x r— x r— x      1    884736 hp-ux*
—rw x r— x r— x      1    884736 SYSBCKUP*
—rw x r— x r— x      1   1032192 hp-ux.test*
```

The contents of the root directory (/) on the root disc 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**

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 pathes 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

a.out(4), boot(1M), errno(2), fsck(1M), init(1M), inittab(4), isl(1M), magic(4), pdc(1M).

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 a user may 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 the 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* will reprompt 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 may enter only one of the digits **0** through **6** or the letters **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**, saying where the logical system console has been relocated.

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 will be rejected and the user will be re-prompted. 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 will terminate 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 level*s 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 will execute before *init* can scan *inittab* and perform the specified functions.

When *init* is requested to change *run level*s (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 *setpgrp*2(2), it will not receive these signals. (Common example 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,q tells *init* to re-examine the **/etc/inittab** file.

s,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 the super-user.

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, and 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 that are 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* will not 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

insf [**–f** *devfile*] [**–N** *cnode*] [**–n** *npty*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d cn**
insf [**–f** *devfile*] [**–N** *cnode*] **–d diag0**
insf [**–f** *devfile*] [**–N** *cnode*] **–d diaghpib1** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d disc0** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d disc1** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d disc2** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d display0** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d dmem**
insf [**–f** *devfile*] [**–N** *cnode*] **–d gpio0** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d gpio1** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d instr0** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d lan0** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d lan1** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d lpr0** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d lpr1** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d meas_drivr**
insf [**–f** *devfile*] [**–N** *cnode*] **–d mm**
insf [**–f** *devfile*] [**–N** *cnode*] **–d mux0** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d mux2** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d osi0** [**–l** *lu*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d pty0** [**–n** *npty*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d pty1** [**–n** *npty*]
insf [**–f** *devfile*] [**–N** *cnode*] **–d scc1**
insf [**–f** *devfile*] [**–N** *cnode*] **–d sw**
insf [**–f** *devfile*] [**–N** *cnode*] **–d sy**
insf [**–f** *devfile*] [**–N** *cnode*] **–d tape1** [**–l** *lu*]

DESCRIPTION

Insf installs special files in the current directory. The **–f** option specifies *devfile*, which is a file that describes drivers and pseudo-drivers. This file is generated by *uxgen*(1M). If the **–f** option is not given, the file **/etc/devices** is used.

If the **–d** option is not entered, *insf* installs special files for every entry in *devfile*. The **–d** option specifies that only special files for a particular driver are to be installed. If the **–l** option is given, special files for only that logical unit will be installed.

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. If it is given, only *npty* pty special files will be installed; if it is omitted, 48 will be installed.

The file permissions are set by *insf*. Unless otherwise noted, the owner and group ID are set to *bin*.

The following sections show which special files are created and their permissions for each driver:

CN

syscon	rw- -w- -w-
systty	rw- -w- -w-
console	rw- -w- -w-

DIAG0

diag0	rw- --- ---

DIAGHPIB1

For each logical unit, the following special files are installed:

diag/hpib/hp28650A/*lu* rw- --- ---

DISC0

Special file names for **disc0** use the following format: c<*lu*>d<*unit*>s<*section*>. For each logical unit, the following special files are installed:

dsk/c<*lu*>d0s<*section*>
: sections 0 to 15, group sys, block entry, rw- r-- ---

rdsk/c<*lu*>d0s<*section*>
: sections 0 to 15, group sys, character entry, rw- r-- ---

ct/c<*lu*>d<*unit*>s2
: units 0 and 1, block entry, rw- rw- rw-

rct/c<*lu*>d<*unit*>s2
: units 0 and 1, character entry, rw- rw- rw-

diag/dsk/c<*lu*>d<*unit*>
: units 0 and 1, character entry, rw- --- ---

DISC1

Special file names for **disc1** use the following format: c<1000+*lu*>d<*unit*>s<*section*>. For each logical unit, the following special files are installed:

dsk/c<1000+*lu*>d0s<*section*>
: sections 0 to 15, group sys, block entry, rw- r-- ---

rdsk/c<1000+*lu*>d0s<*section*>
: sections 0 to 15, group sys, character entry, rw- r-- ---

ct/c<1000+*lu*>d<*unit*>s2
: units 0 and 1, block entry, rw- rw- rw-

rct/c<1000+*lu*>d<*unit*>s2
: units 0 and 1, character entry, rw- rw- rw-

diag/dsk/c<1000+*lu*>d<*unit*>
: units 0 and 1, character entry, rw- --- ---

DISC2

Special file names for **disc2** use the following format: c<2000+*lu*>d<*unit*>s<*section*>. For each logical unit, the following special files are installed:

dsk/c<2000+*lu*>d0s<*section*>
: sections 0 to 15, group sys, block entry, rw- r-- ---

rdsk/c<2000+*lu*>d0s<*section*>
: sections 0 to 15, group sys, character entry, rw- r-- ---

diag/dsk/c<2000+*lu*>d0
: character entry, rw- --- ---

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–
ttyi<*lu*>	rw– –w– –w–
diag/crt<*lu*>	rw– ––– –––

DMEM

dmem	rw– ––– –––

GPIO0

For each logical unit, the following special files are installed:

gpio<*lu*>	rw– rw– rw–
diag/gpio<*lu*>	rw– ––– –––

GPIO1

For each logical unit, the following special files are installed:

gpio<1000+*lu*>	rw– rw– rw–
diag/gpio<1000+*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

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–

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<*lu*>p<*port*>	ports 0 to 5, direct connect, rw– –w– –w–
mux<*lu*>	rw– ––– –––
diag/mux<*lu*>	rw– ––– –––

MUX2

For each logical unit, the following special files are installed:

tty<*lu*>p<*port*>	ports 0 to 7, direct connect, rw– –w– –w–
mux<*lu*>	rw– ––– –––

diag/mux<*lu*> rw– ––– –––

OSI0

For each logical unit, the following special files are installed:

osi<*lu*> rw– rw– rw–
diag/osi<*lu*> rw– ––– –––

PTY0

pty<*index*><*number*> indices 'p' to 'r', numbers 0 to f (hexadecimal), rw– rw– rw–
ptym/pty<*index*><*number*> indices 'a' to 'c' and 'e' to 'z', numbers 0 to f (hexadecimal), rw– rw– rw–
ptym/pty<*index*><*number*> indices 'a' to 'c' and 'e' to 'z', numbers 00 to 99, rw– rw– rw–

The special files **pty[pqr][0-f]** are linked to the special files of the same name in the **ptym** directory.

PTY1

tty<*index*><*number*> indices 'p' to 'r', numbers 0 to f (hexadecimal), rw– rw– rw–
pty/tty<*index*><*number*> indices 'a' to 'c' and 'e' to 'z', numbers 0 to f (hexadecimal), rw– rw– rw–
pty/tty<*index*><*number*> indices 'a' to 'c' and 'e' to 'z', numbers 00 to 99, rw– rw– rw–

The special files **pty[pqr][0-f]** are linked to the special files of the same name in the **ptym** directory.

SCC1

tty<*port*> ports 'a' and 'b', direct connect, rw– –w– –w–

SW

swap group sys, rw– r–– –––

SY

tty rw– rw– rw–

TAPE1

For each logical unit, the following special files are installed:

mt/<*lu*>l 800 bpi, block entry, rw– rw– rw–
mt/<*lu*>m 1600 bpi, block entry, rw– rw– rw–
mt/<*lu*>h 6250 bpi, block entry, rw– rw– rw–

mt/<*lu*>ln no rewind, 800 bpi, block entry, rw– rw– rw–
mt/<*lu*>mn no rewind, 1600 bpi, block entry, rw– rw– rw–
mt/<*lu*>hn no rewind, 6250 bpi, block entry, rw– rw– rw–

rmt/<*lu*>l 800 bpi, character entry, rw– rw– rw–
rmt/<*lu*>m 1600 bpi, character entry, rw– rw– rw–
rmt/<*lu*>h 6250 bpi, character entry, rw– rw– rw–

rmt/<*lu*>hc 6250 bpi, compressed, character entry, rw– rw– rw–

rmt/<*lu*>ln no rewind, 800 bpi, character entry, rw– rw– rw–
rmt/<*lu*>mn no rewind, 1600 bpi, character entry, rw– rw– rw–
rmt/<*lu*>hn no rewind, 6250 bpi, character entry, rw– rw– rw–

diag/mt/<*lu*> character entry, rw– ––– –––

WARNINGS

Insf should only be run in single-user mode. It may 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

/etc/devices

SEE ALSO

lssf(1M), mknod(1M), mksf(1M), uxgen(1M).

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* will search 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 will be searched before the directories specified in the default list.

The meanings of the options are:

–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. May 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. May be used alone or with the **–o** or **–s** options.

–i Ignores default directory list, searching only through the given directories (*dirx ...*). May 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. May 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. May 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 the super-user. May be used alone or with any other option.

–u *user* Causes *file* to be owned by user *user*. This option is available only to the super-user. May be used alone or with any other option.

–s Suppresses printing of messages other than error messages. May 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 in order 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** then the default directory list is further scanned in order to determine where *file* is to be located.

Cross Generation

The environment variable ROOT will be used to locate the locations file (in the form **$ROOT/etc/syslist**). This is necessary in the cases where cross generation is being done on a production system. Furthermore, each pathname in **$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.

SEE ALSO

cpset(1M), make(1).

BUGS

Install cannot create alias links for a command (for example, *vi*(1) is an alias link for *ex*(1)).

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), hpuxboot(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 will be unbusied 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*(1), and thus may not be able to perfectly identify which processes to signal.

If no *signal* is specified, a default of **9** (kill) is used.

Killall is invoked automatically by *shutdown*(1M). The use of *shutdown* is recommended over using *killall* by itself.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), kill(1), ps(1), shutdown(1M), signal(5).

STANDARDS CONFORMANCE

killall: SVID2

NAME

last, lastb – indicate last logins of users and teletypes

SYNOPSIS

/etc/last [*–c*] [*–count*] [*name ...*] [*tty ...*]
/etc/lastb [*–c*] [*–count*] [*name ...*] [*tty ...*]

DESCRIPTION

Last looks back in the *wtmp* file that records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes 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** would list all of root's sessions as well as all sessions on the console terminal. *Last* prints the sessions of the specified users and teletypes, most recent first, indicating when the session began, the duration of the session, and the teletype 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 with no arguments prints a record of all logins and logouts in reverse order. The **–c** option displays login information for all cnodes of an HP Cluster (see the Glossary). The *–count* option limits the report to *count* lines. When used in conjunction with the **–c** option, the limit is applied independently for each cnode of the cluster.

If *last* is interrupted, it indicates how far the search has progressed in *wtmp*. If interrupted with a quit signal (generated by a control-\), *last* indicates how far the search has progressed, then continues the search.

Lastb looks back in the *btmp* database to display bad login information, and is normally executable only by superuser because the *btmp* file 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

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 may only be executed by the super-user.

RETURNS

0 - successful *link*.
1 - input syntax error.
2 - *link* or *unlink* call failed.

SEE ALSO

rm(1), link(2), unlink(2).

EXTERNAL INFLUENCES

International Code Set Support

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

STANDARDS CONFORMANCE

link: SVID2
unlink: SVID2

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* may not 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 then 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, then the class will be deleted, too. No other *options* are allowed with **–x**.

–d[*dest*] Makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when *lpsched*(1M) is running. No other *options* are allowed with **–d**.

The following *options* are only useful with **–p** and may appear in any order. For ease of discussion, the printer will be referred to as printer *P* below.

–c*class* Inserts printer *P* into the specified *class*. *Class* will be 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 supplied.

–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, *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*, then the *class* will be 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, then *lpadmin* may be used while the scheduler is running.

The following *options* are only useful with **−p** and may 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 not to use three-digit request numbers.

−oci*remcancel* Specifies that the local command *remcancel* is used to cancel requests to remote printers. To ensure 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 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 supplied. Only one of the **−e**, **−i** or **−m** options may be supplied. The **−h** and **−l** key letters are mutually exclusive. Printer and class names may be no longer than 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 the directory **/usr/spool/lp/model** and can be used without modification with **lpadmin −m**. All cancel models reside in the directory **/usr/spool/lp/cmodel** and can be used without modification with **lpadmin −ocm**. All status models reside in the 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**. Alternatively, LP administrators can modify copies of models and 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 returned by *lp*.

user logname of the user who made the request.

title optional title specified with the **−t** option of *lp*.

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*. Options from a BSD system have the character sequence **BSD** attached to the beginning of the option (e.g., **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 the printing is completed, it is the responsibility of the interface program to exit with a code indicative of the success of the print job. A return value of **0** indicates that the job completed successfully. Values of **1** through **127** indicate that some error was encountered. This problem will not affect future print jobs. *Lpsched* notifies users by mail that there was an error in printing the request. When problems are detected which are likely to affect future print jobs, the interface program would be well to disable the printer so that print requests are not lost.

The cancel and status model interface programs do the actual communication with the remote system to cancel requests or get status of requests. See *rcancel*(1M) and *rlpstat*(1M) for command line arguments.

In the HP Clustered environment, all printers are attached to the root server and all spooling is handled as if the cluster were a single system. Remote spooling applies to spooling from or to machines outside of the cluster.

EXAMPLES

Assuming an existing Hewlett-Packard HP 2934A line printer named **lp2**, use the **hp2934a** model interface through **/dev/lp** after the command as follows:

/usr/lib/lpadmin −plp2 −mhp2934a −v/dev/lp

Assuming a printer **lp** on a remote system **system**, the command:

/usr/lib/lpadmin −plp3 −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.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1), lpstat(1), nroff(1), accept(1M), lpana(1M), lpsched(1M), mklp(1M), rcancel(1M), rlp(1M), rlpdaemon(1M), rlpstat(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, *lpadmin* behaves as if all internationalization variables are set to "C". See *environ*(5).

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.

One option is supported:

–**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 of waiting time to start to print after being queued into spooler.

Wait SD
: Standard Deviation for waiting time.

Print AV
: Average of printing time to have been completed after being started to print.

Print SD
: Standard Deviation for printing time.

Bytes AV
: Average of number of bytes to being printed per request.

Bytes SD
: Standard Deviation for number of bytes.

Sum KB
: Sum of bytes to have being printed for all requests in Kbytes.

Num of Requests
: Total number of requests from logging started.

In the HP Clustered environment, all printers are attached to the root server and all spooling is handled as if the cluster were a single system. Remote spooling applies to spooling from or to machines outside of the cluster.

WARNINGS

Lpana performs its operation on the local system (or HP cluster) only.

AUTHOR

Lpana was developed by the Hewlett-Packard Company.

FILES

/usr/spool/lp/lpana.log

SEE ALSO

lp(1), lpstat(1). lpadmin(1M), lpsched(1M).

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.

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/iib/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**.

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 will stop printing. Requests that were printing at the time a printer was shut down will be 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 may 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) will reject 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 *priority* for which spooled file needs to be printed. *Fence* must be in between 0 (lowest fence) and 7 (highest fence). Each *printer* has its own *fence*, and is initialized to 0 when it is configured by *lpadmin*(1M) command. *Lpfence* is used only when *lpsched* is not running.

In the HP Clustered environment, all printers are attached to the cluster server and all spooling is handled as if the cluster were a single system. Remote spooling applies to spooling from or to machines outside of the cluster.

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).

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).

NAME

lsdev – list device drivers in the system

SYNOPSIS

/etc/lsdev [major...]

DESCRIPTION

With no arguments, *lsdev* lists, one pair per line, the major device numbers and driver names of all 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 there are any arguments, they must represent major device numbers. For each, *lsdev* lists the corresponding driver name (if any).

Lsdev is simply a quick-reference aid. In some implementations, it may only read an internal list of device drivers, not the actual list in the operating system.

DIAGNOSTICS

Lists the drivername as "NO SUCH DRIVER" when appropriate.

WARNINGS

Some device drivers available from the system may be intended for use by other drivers. You may get unexpected results if you try to use them directly from a special file.

AUTHOR

Lsdev was developed by HP.

SEE ALSO

Pages on specific device drivers in section (7).

NAME

lssf – list a special file

SYNOPSIS

lssf [-f devfile**]** path...

DESCRIPTION

Lssf lists a special file. The **-f** option specifies *devfile*, which is a file that describes drivers and pseudo-drivers. This file is generated by *uxgen*(1M). If the **-f** option is not present, then the file **/etc/devices** is used.

For each path, *lssf* determines the major number of the special file and whether it is block or character (using *stat*(2)). *Lssf* scans *devfile* for the driver which matches the major number of the special file. When the driver is found, the minor number of the special file is decoded and a mnemonic description is printed on standard output. The mnemonics used to describe the minor number fields are closely related to the options used with *mksf*(1M) which makes a special file.

As an example, suppose a special file is created with the command "mksf -d disc0 -l 1 -u 2 -s 3 dsk/c1d2s3". The command "lssf dsk/c1d2s3" will output "disc0 lu 1 unit 2 section 3 dsk/c1d2s3".

AUTHOR

Lssf was developed by HP.

FILES

/etc/devices

SEE ALSO

mksf(1M), insf(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 mentioned (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

The following options are supported:

-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 those 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**, since these 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 to fully recover from a failure.

Many messages only make sense with an understanding of how the program works. In particular, a temporary file is used to hold *file* while the CDF is created, then the temporary file is 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.

DEPENDENCIES

Series 800

Access control lists are not implemented.

RFA and 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(5), context(5), clusterconf(4), acl(5).

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.

There must be exactly one of the options **-c**, **-u**, **-o**, **-r**, and **-l**.

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, and 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 may not 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).

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, then 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.

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.

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.

mirror -l [*device* ...]

List mirrors. With no devices named, all configured mirrors are listed. Otherwise, mirrors containing the given devices are listed.

The output is 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, then 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.

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 diskless client of a clustered system.

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

HP 9000 Series 800 with 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 the 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, like mail notification or logging, in the case of disk failures.

Mirrorlog can be executed only by the super-user.

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 Hewlett-Packard.

FILES

/dev/rdsk/mirconfig	access to mirror driver
/etc/mirrortab	log file

SEE ALSO

mirror(1m), mirrortab(4).

NAME

mkdev – make device files

SYNOPSIS

/etc/mkdev

DESCRIPTION

This shell script helps the superuser install and maintain an HP-UX system. It consists of a machine-dependent list of commands which 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.

This command makes it easier to build (or rebuild) special files all at once.

Mkdev automatically changes the working directory (using *cd*) 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. You may want to use shorter device names than those suggested, 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.

Since the super-user must modify this script before using it the first time, an error is given if it has not been modified.

AUTHOR

Mkdev was developed by the Hewlett-Packard Company.

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs [**–L** | **–S**] *special size* [*nsect ntrack blksize fragsize ncpg minfree rps nbpi*]
/etc/mkfs [**–L** | **–S**] *special proto* [*nsect ntrack blksize fragsize ncpg minfree rps nbpi*]

REMARKS

HFS file systems are normally created with the *newfs*(1M) command.

DESCRIPTION

Mkfs constructs a file system by writing on the special file *special*. The *size* specifies the number of DEV_BSIZE blocks in the file system. (DEV_BSIZE is defined in **<sys/param.h>**.) *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.

The optional arguments allow fine tune control over the parameters of the file system.

Nsect specifies the number of sectors per track on the disk.

Ntrack specifies the number of tracks per cylinder on the disk.

Blksize gives the primary block size for files on the file system. It must be a power of two, currently selected from 4096 or 8192.

Fragsize gives the fragment size for files on the file system. The **fragsize** represents the smallest amount of disk space that will be allocated to a file. It must be a power of two currently selected from the range DEV_BSIZE to MAXBSIZE.

Ncpg specifies the number of disk cylinders per cylinder group. This number must be in the range 1 to 32.

Minfree specifies the minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the super-user is allowed to allocate disk blocks. The default value is 10%. If a disk does not revolve at 60 revolutions per second, the **rps** parameter may be specified.

The **nbpi** argument specifies the 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, its value defaults to 2048.

The **-L** or **-S** option specifies that *mkfs* should build a file system that allows directory entries (file names) to be up to MAXNAMLEN (255) or DIRSIZ (14) bytes long, respectively. There are two types of HFS file systems; they are distinguished mainly by differing directory formats that place these different limits on the length of file names. If neither option is specified, *mkfs* creates a file system of the same type as the root file system.

If the second argument is a file name that can be opened, mkfs 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 new lines. 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 "" then 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, 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 *chmod*(1).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence 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, *mkfs* 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:

```
/etc/BOOT
4872
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
        $
$
```

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, you can run *bdf* **-i** on existing file systems. For more information on access control lists, see *acl*(5).

WARNINGS

No way to specify links in the **proto** file.

DEPENDENCIES

Series 800

Access control lists are not implemented.

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), acl(5).

NAME

mklost+found – make a lost+found directory for fsck

SYNOPSIS

/etc/mklost+found

DESCRIPTION

Mklost+found creates a directory *lost+found* in the current directory. It also creates several empty files, which are then removed to provide empty slots for *fsck*(1M).

For the HFS file system, *mklost+found* is not normally needed, since *mkfs*(1M) automatically creates the *lost+found* directory when a new file system is created.

AUTHOR

Mklost+found was developed by the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

fsck(1M), mkfs(1M).

NAME

mklp – configure the LP spooler subsystem

SYNOPSIS

/etc/mklp

REMARKS

This command is implemented as a shell script, and differs for various HP-UX implementations. This description applies to all systems, and further details are found in the commentary of the script itself.

DESCRIPTION

This shell script helps the superuser configure into the LP spooler the printers and plotters that are supported on the particular HP-UX system. Administration of all printers and plotters in the LP spooler subsystem is similar, although available printing or plotting options supported by specific devices vary, depending on the model. These options are described within the *mklp* script itself.

This command makes it easier to configure the LP spooler as a single operation. If desired, *mklp* can also be used to rebuild the subsystem.

Mklp should be modified before each use. Command lines for printers that are not being used should be commented out with **#** to keep them available for later use.

DIAGNOSTICS

Each command line in *mklp* is echoed as it is executed. Error messages, if any, are generated by the commands invoked.

Since the superuser must modify this script before using it for the first time, *mklp* returns an error message if it has not been modified.

DEPENDENCIES

Series 300

While the *mklp* script indicates how the device special files should be defined, the *mkdev*(1) script should also be used to determine the major and minor number.

AUTHOR

Mklp was developed by HP.

SEE ALSO

chmod(1), mknod(1M).

NAME

mknod – create special and FIFO files

SYNOPSIS

/etc/mknod *name* **c** | **b** *major minor* [*cnode_name*]
/etc/mknod *name* **p**
/etc/mknod *name* **n** *nodename*

DESCRIPTION

Mknod makes a directory entry and corresponding inode for following special files:

- Device special file (first SYNOPSIS line form),
- FIFO file, sometimes called a named pipe (second SYNOPSIS line form),
- Network special file (third form). See NETWORKING FEATURES below.

Name is the path name of the special file to be created.

Device Special Files

For device special files, the second argument should be:

b for block-type special files (such as disks), or

c for character-type special files (other devices such as 9-track tapes or line printers).

Other arguments:

major number specifying major device type (e.g., device driver number)

minor number specifying the device location (typically, but not exclusively, the unit, drive, HP-IB bus address and/or line number).

cnode_name, if present, specifies the cnode name or cnode ID (see *glossary*(9)). A call to *mkrnod()* is made (see *mknod*(2)), with cnode ID determined as follows:

- If *cnode_name* is a numeric value, it is accepted as the cnode ID.
- If *cnode_name* is non-numeric, it is interpreted as the cnode name, and file **/etc/clusterconf** is searched to determine the cnode ID.

Assignment of major and minor device numbers is specific to each HP-UX system. *Major* and *minor* can each be specified in hexadecimal, octal, or decimal, using C language conventions (decimal numbers: no leading zero, octal numbers: leading zero, hexadecimal: leading zero followed by 'x'.) Requires real user ID of 0 (super-user).

Named Pipes

To create named pipes or FIFO buffer files, use **p** as the second argument to *mknod*. [The *mkfifo*(1) command can also be used for this purpose.] All users can use *mknod* to create named pipes.

Permissions

The newly created file has a mode of 0666, as modified by the current setting of the user's *umask*.

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.

NETWORKING FEATURES

RFA

Mknod is also used to create a network special file. A network special file addresses another node identified by *nodename* on a local area network. Requires real user ID of 0 (super-user).

SEE ALSO

chacl(1), mkfifo(1) lsdev(1M), mknod(2), mknod(4), acl(5).
HP-UX System Administrator Manual

STANDARDS CONFORMANCE

mknod: SVID2, XPG2

NAME

mkpdf – create a Product Description File from an prototype PDF

SYNOPSIS

mkpdf [**–c** *comment_string*] [**–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 *version, checksum, size,* and *linked_to* target fields are forced to be empty. If the file is a device, the *version, checksum, size,* and *linked_to* fields are forced to be empty and the *size* field contains the major and minor device numbers. If a pathname is prefaced with "**?**", the file is assumed to be an optional file. This file will be processed like all other files except that if the file does not exist, values provided in the prototype will be reproduced.

If – is used for either the *prototype_PDF* or *new_PDF*, the command assumes that stdin and/or stdout is being used for the approprate value.

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.

–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 NULL.

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*. The **–r** option specifies that a root of */PRODUCT* should be prefixed to each pathname while the directory is being searched. The – in the prototype PDF position specifies that *stdin* is being used for the prototype PDF file. 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 -C -r /PRODUCT - PDF
```

WARNINGS

Sizes reported do not reflect blocks allocated to directories (including CDFs).

AUTHOR

Mkpdf was developed by Hewlett-Packard Company.

SEE ALSO

pdfck(1M), pdfdiff(1M), pdf(4).

NAME

mkrs – construct a recovery system

SYNOPSIS

/etc/mkrs [**–v**] [**–f** *rcdev*] [**–r** *rootdev*] [**–m** *series*]

DESCRIPTION

Mkrs constructs a recovery system on removable media. 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

The following *options* are recognized:

–v Normally, *mkrs* does its work silently. The **v** (verbose) option prints a running history of the construction process.

–f *rcdev* Specify the name of the device file for the recovery system (that is, the cartridge tape drive or alternate hard disc 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 then 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 then 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.

An error message results if none of the default device files for the root device exist and the **–r** option is not used to specify a root device file.

An error message results if 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.

AUTHOR

Mkrs was developed by HP.

SEE ALSO

HP-UX System Administrator Manual

config(1M), mkfs(1M).

NAME

mksf – make a special file

SYNOPSIS

mksf [**-f** *devfile*] **-d disc0** [**-l** *lu*] [**-u** *unit*] [**-s** *section*] [**-c**] [**-r**] [**-t**] [*path*...]

mksf [**-f** *devfile*] **-d disc1** [**-l** *lu*] [**-u** *unit*] [**-s** *section*] [**-c**] [**-r**] [**-t**] [*path*...]

mksf [**-f** *devfile*] **-d disc2** [**-l** *lu*] [**-u** *unit*] [**-s** *section*] [**-r**] [**-t**] [*path* ...]

mksf [**-f** *devfile*] **-d framebuf** [**-l** *lu*] [**–t**] [*path*...]

mksf [**-f** *devfile*] **-d gpio0** [**-l** *lu*] [**-t**] [*path* ...]

mksf [**-f** *devfile*] **-d hil** [**-l** *lu*] [**-a** *address*] [*path* ...]

mksf [**-f** *devfile*] **-d hilkbd** [**-l** *lu*] [*path* ...]

mksf [**-f** *devfile*] **-d instr0** [**-l** *lu*] [**-a** *address*] [**-r**] [**-t**] [*path* ...]

mksf [**-f** *devfile*] **-d ite** [**-l** *lu*] [*path* ...]

mksf [**-f** *devfile*] **-d lpr0** [**-l** *lu*] [**-c**] [**-n**] [**-o**] [**-e**] [**-r**] [**-t**] [*path* ...]

mksf [**-f** *devfile*] **-d lpr1** [**-l** *lu*] [**-c**] [**-n**] [**-o**] [**-e**] [**-r**] [**-t**] [*path* ...]

mksf [**-f** *devfile*] **-d mux0** [**-l** *lu*] [**-p** *port*] [**-h|-i|-o**] [**-c**] [*path* ...]

mksf [**-f** *devfile*] **-d mux1** [**-l** *lu*] [*path* ...]

mksf [**-f** *devfile*] **-d pseudo** [**-m** *minor*] *path* ...

mksf [**-f** *devfile*] **-d tape1** [**-l** *lu*] [**-b** *bpi*] [**-a|-u**] [**-c**] [**-n**] [**-r**] [**-t**] [**-w**] [**–C**] [*path* ...]

DESCRIPTION

Mksf makes a special file. The **-f** option specifies *devfile*, which is a file that describes drivers and pseudo-drivers. This file is generated by *uxgen*(1M). If the **-f** option is not present, then the file **/etc/devices** is used. The **-d** option specifies the driver name. Other options depend on the driver name.

Mksf scans *devfile* to determine the major number of the driver. While scanning *devfile*, the *lu* (logical unit) is checked for validity. Hence, a special file cannot be created for a device that is not in the *devfile*.

The following arguments are common to many drivers:

-l The logical unit number (*lu*) of a device. The *lu* is assigned by *uxgen*(1M).

-t Transparent mode (normally used by diagnostics).

path Name of the special file. *Path* is created in the current directory. If the file already exists, it will be removed and then created. Most drivers have a default name for *path*.

Each driver also has a specific set of arguments, which are shown in the following sections.

DISC0

-c This argument must be present if the unit is a cartridge tape.

-u *unit* The cs80 unit number (for example, unit 0 - disc, unit 1 - tape).

-r Raw; create character, not block, special file.

-s *section* The section number.

path The default *path* name depends on the arguments **-r** and **-c**:

rct/c*lu* **d***unit* **s***section*,	if **-r** and **-c**
rdsk/c*lu* **d***unit* **s***section*,	if **-r** and not **-c**
ct/c*lu* **d***unit* **s***section*,	if not **-r** and **-c**
dsk/c*lu* **d***unit* **s***section*,	if not **-r** and not **-c**

DISC1

-c This argument must be present if the unit is a cartridge tape.

-u *unit* The cs80 unit number (for example, unit 0 - disc, unit 1 - tape).

-r Raw; create character, not block, special file.

-s *section* The section number.

path The default *path* name depends on the arguments **-r** and **-c**:

rct/c 1000+ *lu* **d***unit* **s***section*,	if **-r** and **-c**
rdsk/c 1000+ *lu* **d***unit* **s***section*,	if **-r** and not **-c**
ct/c 1000+ *lu* **d***unit* **s***section*,	if not **-r** and **-c**
dsk/c 1000+ *lu* **d***unit* **s***section*,	if not **-r** and not **-c**

DISC2

-u *unit* The cs80 unit number (typically 0).

-r Raw; create character, not block, special file.

-s *section* The section number.

path The default *path* name depends on the argument **-r**:

rdsk/c 2000+ *lu* **d***unit* **s***section*,	*if* **-r**
dsk/c 2000+ *lu* **d***unit* **s***section*,	*if not* **-r**

FRAMEBUF

path The default path name is **crt***lu*.

GPIO0

path The default path name is **gpio***lu*.

HIL

-a *address* The link address (1-7).

path The default path name is **hil_***lu.address*.

HILKBD

path The default path name is **hilkbd***lu*.

INSTR0

-a *address* The HP-IB instrument address (0-30).

-r Raw; the special file has no associated HP-IB instrument address.

path The default path name is **hpib/***lu***a***address* or **hpib/***lu* (if **-r**).

ITE

path The default path name is **tty*i******lu***.

LPR0/LPR1

-c Caps. 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.

path default path name is **lp***lu* or **rlp***lu* (if **-r**).

MUX0

-c CCITT.

-h Hardwired (direct connect).

-i Callin.

-o Callout.

-p *port* Multiplexor port number (0-5).

path The default path name is **tty***lu***p***port*

MUX1

path Default path name is **mux***lu*.

PSEUDO (cn/diag0/dmem/meas_drivr/mm/pty0/pty1/sw/sy)

-m *minor* The minor number.

path No default. Path must be specified.

TAPE1

-a AT&T style rewind/close.

-b *bpi* Bits per inch. Values of *bpi* are **800**, **1600**, **6250**.

-c RTE compatible close.

-n No rewind on close.

-r Raw; create character, not block, special file.

-u UC Berkeley style rewind/close.

-w Wait (disable immediate reporting).

−C Enable data compression.

path The default path 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.

AUTHOR

Mksf was developed by HP.

FILES

/etc/devices

SEE ALSO

insf(1M), lssf(1M), uxgen(1M).

NAME

mount, umount – mount and unmount file system

SYNOPSIS

/etc/mount [*fsname directory* [**−frv**] [**−o** *options*] [**−t** *type*]]
/etc/mount -a [**−fv**] [**−t** *type*]
/etc/mount -p

/etc/umount [**−v**] *fsname*
/etc/umount [**−v**] *directory*
/etc/umount -a [**−v**] [**−h** *host*] [**−t** *type*]

DESCRIPTION

The *mount* command announces to the system that a removable file system is to be attached to the file tree at *directory*. The *directory* must exist already; it becomes the name of the root of the newly mounted file system. *Directory* must be given as an absolute path name. *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.

The *umount* command 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 may be specified.

In the HP Clustered environment, only NFS file systems can be mounted and unmounted from client nodes. See "Networking Features", below.

Options (*mount*)

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*(1M) run on it before mounting.

−p Print the list of mounted file systems in a format suitable for use in **/etc/checklist**.

−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 will occur when access times are updated, whether or not any explicit write is attempted.

−t *type* Specify a file system *type*. The accepted 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 to a specific *type* only.

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.

Options (*umount*)

−a Unmount all files described in **/etc/mnttab**.

−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:

bg	If the first *mount* attempt fails, retry in the background.
fg	Retry in foreground (default).
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).
wsize=*n*	Set write buffer size to *n* bytes (default set by kernel).
timeo=*n*	Set NFS timeout to *n* tenths of a second (default = 7).
retrans=*n*	Set number of NFS retransmissions to *n* (default = 4).
port=*n*	Set server IP port number to *n* (default is the port customarily used for NFS servers).
soft	Return error if server does not respond.
hard	Retry request until server responds (default).
intr	Permit interrupts for hard mounts (default).
nointr	Ignore interrupts for hard mounts.
devs	Allow access to local devices (default).
nodevs	Deny access to local devices.

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 time-out 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 will not be interruptable. The **retry** requests will 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.

DIAGNOSTICS

Attempts to mount a currently mounted non-**nfs** volume under another name will result 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.

EXAMPLES

mount /dev/dsk/c0d0s4 /usr Mount a local disk.

WARNINGS

Some degree of validation is done on the file system; however it is generally unwise to mount garbage file systems.

NFS

EXAMPLES

mount −t nfs serv:/usr/src /usr/src
Mount remote file system.

mount serv:/usr/src /usr/src
Same as above.

mount −o soft,ro serv:/usr/src /usr/src
Same as above but with a soft mount; the file system is mounted read-only.

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), mount(2), mnttab(4), checklist(4).

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, lI 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 descendant 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 the super-user can use *mvdir*.

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** will report an error, but **mvdir x/y ./x/y/z/t** (effectively the same command) will not, and a cyclic sub-tree will result. The super-user is cautioned to 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

mkdir(1), cp(1).

EXTERNAL INFLUENCES

International Code Set Support

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

STANDARDS CONFORMANCE

mvdir: SVID2

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.

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.

DIAGNOSTICS

When the file system structure is improper, **??** denotes the "parent" of a parentless file and a path-name beginning with ... denotes a loop.

AUTHOR

Ncheck was developed by AT&T and HP.

SEE ALSO

sort(1), fsck(1M), cdf(4), checklist(4), acl(5).

EXTERNAL INFLUENCES

International Code Set Support

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

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**]

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 host system to a client host system. Currently, the *update*(1M) utility is available as a netdist client.

Superuser privilege is not required to run *netdistd*.

Options

The options are:

–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).

–l Append event information to the default log file (see FILES below). *Netdistd* does no logging unless this option or **–L** is specified. *Netdistd* seeks to the end of the log file before each write to it, so it is 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 Internet 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** multiple times increases the relative amount of verbosity.

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 *netdistd* itself and contain its process ID number. 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. *Netdistd* starts a new child process to handle each connection (service request).

Control Signals

Once *netdistd* is running, you can send it a signal using *kill*(1) to change its state:

SIGHUP Re-read the central package definition file. Also, if logging is enabled, close and reopen the log file.
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* driver 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.

Setting Up a Network Distribution Server

When setting up a system to support the netdist service, follow these steps:

1. Use *updist*(1M) 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: *sysrm*(1M) does not know how to 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.

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. *Netdistd* 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*(1M) directly, because it appears to terminate immediately, possibly causing *init* to respawn it.

4. The server system is now ready for use by *update*(1M). 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**

Security

Whenever the *netdistd* daemon receives a connection request, it performs a security check using the same mechanism as *inetd*(1M); see *inetd.sec*(5).

RETURN VALUE

Netdistd returns 0 if it encounters no errors, and 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 standard error 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 2 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

AUTHOR

Netdistd was developed by HP.

FILES

/netdist/MAIN.pkg	default central package definition file
/netdist/300/*fileset-name***/netdist.pkg**	Series 300 package definition files
/netdist/800/*fileset-name***/netdist.pkg**	Series 800 package definition 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), sysrm(1M), update(1M), services(5), inetd.sec(5).

NAME

newfs – construct a new file system

SYNOPSIS

/etc/newfs [**−L** | **−S**] [**-n**] [**-v**] [*mkfs-options*] *special disk-type*

DESCRIPTION

Newfs is a "friendly" front-end to the *mkfs*(1M) program. *Newfs* will look up the type of disk a file system is being created on in the disk description file */etc/disktab*, calculate the appropriate parameters to use in calling *mkfs*, then build the file system by forking *mkfs* and, if the file system is a root section, install the necessary bootstrap programs in the initial 8192 bytes of the device. The **−n** option prevents the bootstrap programs from being installed. **special** is the character special file for the disk and **disk-type** is the type of the disk as specified in /etc/disktab.

If the **−v** option is supplied, *newfs* will print out its actions, including the parameters passed to *mkfs*.

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 with either the **−L** option, which indicates a file system that allows file names up to MAXNAMLEN (255) bytes long, or with the **−S** option, which indicates a file system that allows file names at most DIRSIZ (14) bytes long.

Options that may be used to override default parameters passed to *mkfs* are:

−s *size* The size of the file system in DEV_BSIZE blocks (defined in <**sys/param.h**>).

−b *block-size* The block size of the file system in bytes.

−f *frag-size* The fragment size of the file system in bytes.

−t *#tracks/cylinder*
The number of tracks per cylinder.

−c *#cylinders/group*
The number of cylinders per cylinder group in a file system. The default value used is 16.

−m *free space %*
The percentage of space reserved from normal users; the minimum free space threshold. The default value used is 10%.

−r *revolutions/minute*
The speed of the disk in revolutions per minute (normally 3600).

−i *number of bytes per inode*
This specifies the density of inodes in the file system. The 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 your need for extra inodes, you can run *bdf* **−i** on existing file systems. For more information on access control lists, see *acl*(5).

FILES

/etc/disktab for disk geometry and file system section information

DEPENDENCIES

Series 800

Newfs will not install bootstrap programs in a root section since the boot programs are kept in a separate section.

Access control lists are not implemented.

AUTHOR

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

SEE ALSO

bdf(1M), fsck(1M), mkfs(1M), tunefs(1M), disktab(4), fs(4), acl(5).

NAME

nlioinit – initialize Native Language I/O

SYNOPSIS

/usr/lib/nlioinit *tty pty tb* [*server*]
/usr/lib/nlioinit **–p** *lp_device pty pb* [*server*]

DESCRIPTION

Nlioinit initializes Native Language I/O by attaching a translating server to a terminal or printer device. If the optional argument **–p** is specified, *nlioinit* assumes that the device is a printer.

The argument *tty* is the name of the terminal device file and *lp_device* is the name of the printer device file in **/dev** that is associated with the user terminal or the printer to which the server will be attached.

The argument *pty* is the name of the master side of a pseudo terminal in **/dev** (for example, **ptyp1**, or **ptym/ptyp1**) that the server will use to perform the translation. Native Language I/O is available to the user process on the slave side of the pseudo terminal whose master is *pty*.

The argument *tb* specifies the terminal behavior for the I/O, and is designated by the following values:

bj	Japanese
bk	Korean
bc	Simplified Chinese
bt	Traditional Chinese
hp35714a	Japanese, with 25 screen lines
atbj	Japanese, with 24 lines and a line used for host input conversion

The **hp35714a** and **atbj** represent HP-16 terminals that require host input conversion.

The argument *pb* specifies the printer behavior for the I/O, and is designated by the following values:

hp41063aj	Japanese
hp41063ac	Simplified Chinese
hp41063at	Traditional Chinese
hpc1200aj	Japanese
hpc1200ak	Korean
hpc1200ac	Simplified Chinese
hpc1200at	Traditional Chinese

The optional argument *server* is the name of the server that *nlioinit* invokes to perform Native Language I/O. If the *server* is omitted, the default server **bserver** for terminal or **pserver** for printer is invoked.

Terminal servers are capable of conversion between internal code (HP-15) and external code, although some features are not available on all servers. When the *stty*(1) command option **icanon** is set, entering a single erase character erases the preceding 16-bit or 8-bit input character. The printer server is also capable of conversion from internal code (HP-15) to external code.

bserver Code conversion between internal code (HP-15) and external code for the Japanese, Korean, Simplified Chinese, and Traditional Chinese languages. It is

not capable of host input conversion.

j0server Code conversion for the Japanese language only. Also capable of host input conversion using word and phrase dictionaries. It uses the input window for 16-bit character input.

pserver Code conversion from internal code (HP-15) to external code for the Japanese, Korean, Simplified Chinese, and Traditional Chinese languages. It provides the same functionality as the line printer driver (see *lp*(7)).

DIAGNOSTICS

Nlioinit complains about fatal errors to **/dev/console**.

EXAMPLES

To initialize Native Language I/O for the terminal on **/dev/tty0p1**, add the following entries to the **/etc/inittab** file:

```
p1:2:respawn:/etc/getty pty/ttyp1 9600
s1:2:respawn:/usr/lib/nlioinit tty0p1 ptym/ptyp1 bj
```

and, execute:

```
init 2
```

WARNINGS

If the specified terminal behavior is invalid or not appropriate to the invoked server, *nlioinit* does not warn the user, and the server works with its default behavior. Eight-bit character I/O is always available. However, sixteen-bit character I/O may be unpleasant if, for example, code conversion is not done by the particular server selected.

Do not set the tty special control characters such as intr, erase, and kill (see *stty*(1)) to ASCII characters with values in the range of **'\021'** to **'\0176'** inclusive for the terminal that the Native Language I/O server is associated with, because HP-15 can include these values as the second byte of a 16-bit character code.

The pty used by the server is allocated upon invocation of this command, and is thereafter made unavailable to other services. The system administrator may wish to consider adding more ptys to the system if many Native Language I/O servers will be used.

AUTHOR

Nlioinit was developed by HP.

SEE ALSO

nlio(1), nlioenv(1), nliostart(1), stty(1), getty(1M), init(1M), mknod(1M), inittab(4), lp(7), pty(7), termio(7).

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.

NAME

opx25 – execute HALGOL programs

SYNOPSIS

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:

–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 the Hewlett-Packard Company.

SEE ALSO

getx25(1), uucp(1).

UUCP, a tutorial in *HP-UX Concepts and Tutorials.*

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)).

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* then provides a 10-second delay, during which time the operator can override the *autoboot* sequence by entering 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 and *autoboot* and *autosearch* enable may be modified via *isl*.

SEE ALSO

boot(1M), hpuxboot(1M), isl(1M).

NAME

pdfck – compare Product Description File to File System

SYNOPSIS

pdfck [**–r** *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 *pdfdiff*(1M). 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

–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.

EXAMPLE

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 [**–p** *percent*] *pdf1 pdf2*

DESCRIPTION

Pdfdiff is a program that compares two PDFs (Product Description Files). The PDFs can be generated using the *mkpdf*(1M) program. 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 will be 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

–p *percent* specifies a threshhold percentage for file growth. Files having a net size change greater than or equal to this percentage will be reported. A decrease in size is reported as a negative number. If **–p** is not specified, a default value of zero percent is used.

EXAMPLE

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

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* checks only the local password and group files. The Yellow Page 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

rbootd – remote boot server

SYNOPSIS

/etc/rbootd [**–a**] [**–l** *loglevel*] **–L** *logfile*] [*landevice*]

DESCRIPTION

Rbootd services initial boot-up requests from an HP Cluster client over a local area network. *Rbootd* only responds to requests from machines that are listed in the cluster configuration file.

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*:

cnode name
architecture
remoteroot
default

where *cnode name* is based on information found in the cluster configuration file and *architecture* is either HP-PA, or HP–MC68881 HP–MC68020 HP–MC68010, based on the identity provided to *rbootd* by the booting client.

Options

Rbootd supports the following options:

–a Append to the *rbootd* log file. By default *rbootd* truncates the log file when it is started.

–l *loglevel* Set the amount of information that will be logged in the log file. *Rbootd* supports the following logging levels:

- **0** Only log startup and termination of *rbootd*.
- **1** Log all errors. This is the default logging level.
- **2** Log rejected boot requests from machines not found in **/etc/clusterconf**.
- **3** Log all boot requests.

–L *logfile* Specifiy 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

SEE ALSO

cdf(4), clusterconf(4), context(5).

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*(1) command.

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

id — Specifying a request ID (as returned by *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*(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, then this option only specifies the printer to perform the cancel operation on.

−a — Remove all requests which a user owns 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 the superuser.

−u *user* — Remove any requests queued belonging to that user (or users). This form of invoking *rcancel* is useful only to the superuser.

In the HP Clustered environment, all printers are attached to the root server and all spooling is handled as if the cluster were a single system. Remote spooling applies to spooling from or to machines outside of the cluster.

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

–h Shutdown the system and halt.

–r Shutdown 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 which 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*(2) with arguments as indicated by the other options. 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, while 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.

–d *device* 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* 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*
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, "0x80009006997". This number must be in a format acceptable to *reboot*(2). 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* Use the 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/boot_servers**. 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 root server of a cluster causes all of the cluster nodes to be rebooted. Each cluster node will be rebooted by a *reboot* command executing locally. The arguments for these local *reboot* commands are copied from the root server's *reboot* command, with the exception of the **−d** and **−f** options. Using the **−l** or **−b** option on the root server causes all members of the cluster, including the root server, to reboot using the new server. On a client system, only the client is rebooted, and the effect is to change root servers.

Executing *reboot* on a client node only affects that node.

Only the super-user can execute the *shutdown* command.

DEPENDENCIES

Series 800

The **−f** option is not supported. The device specified with the **−d** option must refer to a LAN interface.

AUTHOR

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

FILES

/usr/adm/shutdownlog	shutdown log
/etc/boot_servers	table of system names usable with the **−b** option.

SEE ALSO

lif*(1), reboot(2), lif(4).

NAME

regen – regenerate (uxgen) an updated HP-UX system

SYNOPSIS

regen [-I] [-i] [-L] [-N] [-F] [-O] [-M] [-S] [-B] [-X] [-K] [-G] [-c] [-p] [-A]

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 will be commented out of the configuration file if it is there. 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**, moves the new one from **/etc/conf/S800/hp-ux** to **/hp-ux**, and copies the new devices file to **/etc/devices**. If the **–i** option is used, *regen* finally does an *insf*(1M) and copies an **/etc/newconfig/inittab** into **/etc/inittab.**

Options

–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 1.

–i Install mode. An S800 file is always created.

–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* will not modify it according to the options specified on the command line or in the **.regenrc** file.

–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.

–F The NFS product has been loaded. Include the NFS subsystem.

–L The Lan Link product has been loaded. Include the Lan Link subsystem.

–N The NS/9000 Services product has been loaded. Include the NS/9000 Services subsystem.

–M The Mirror Disk product has been loaded. Include the Mirrored Disk subsystem.

–S The Fairshare Scheduler product has been loaded. Include the Fairshare Scheduler subsystem.

–O The OSI Link product has been loaded. Include the OSI Link subsystem.

–B The BX25 product has been loaded. Include the BX25 subsystem.

-X The XT product has been loaded. Include the XT subsystem.

-K The Datakit product has been loaded. Include the Datakit subsystem.

-G The G2 product has been loaded. Include the G2 subsystem.

-A Absolutely no interaction is allowed. Regen exits with a status 1 if it cannot create a kernel without asking the user questions.

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.

AUTHOR

Regen was developed by HP.

FILES

/etc/devices
/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), devices(4).

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 a *rdump*(1M) followed by a *newfs*(1M) and a *restore* or a *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, then the root directory is extracted, which results in the entire content of the tape being extracted, unless **h** has been specified.

t The names of the specified files are listed if they occur on the tape. If no file argument is given, then 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, then 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, then 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 and 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 may be used in addition to the letter that selects the function desired:

b Specify the block size of the tape in kilobytes. 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 -)

In case of *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. Nomally *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 it can. *Restore* and *rrestore* will not ask whether to abort the restore if get a tape error. They will attempt to skip over the bad tape block(s) and continue as best it 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 it gets a read error. If the **y** modifier has been specified, or the user responds "y", *restore* and *rrestore* will attempt to continue the restore.

If the dump extends over more than one tape, *restore* and *rrestore* will ask the user to change tapes. If the **x** or **i** function has been specified, *restore* and *rrestore* will also 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 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". Common errors are given below.

filename: **not found on tape**
The specified file name was listed in the tape directory, but was 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. Because 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	the 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*(1)). Path names begin in column one 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*(1) outputs its results).

For each path name, *revck* checks that the file exists, and that executing *what*(1) 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. If a *ref_file* is not in the right format, you will get unpredictable results.

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 may be set only on the original system. Transfers of a remote request only use 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** to **4**.

–w*num* Take the immediately following number to be 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.

In the HP Clustered environment, all printers are attached to the root server and all spooling is handled as if the cluster were a single system. Remote spooling applies to spooling from or to machines outside of the cluster.

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. It does not hurt to specify unused options.

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, with 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.**

The following entry should exist in **/etc/services** for remote spooling:

```
printer         515/tcp         spooler
```

In the HP Clustered environment, all printers are attached to the root server and all spooling is handled as if the cluster were a single system. Remote spooling applies to spooling from or to machines outside of the cluster.

EXAMPLES

To start *rlpdaemon* from **/etc/rc**, invoke the following 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
```

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

ARPA Services/9000 Series 800 Manual Reference Pages: inetd(1M), hosts.equiv(4), inetd.conf(4), inetd.sec(4), services(4).

ARPA Services/9000 Series 800 Node Manager's Guide.
HP-UX System Administrator Manual.

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. Without any arguments, *rlpstat* reports on any requests currently in the queue.

For each request submitted (i.e., invocation of *lp*(1)) *rlpstat* reports the request ID, user's name, total size of the request, the 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*(1M) command and should not be invoked directly.

Options

–d *printer* — Specify a particular printer. Otherwise, the default line printer is used (or the value of the LPDEST variable in the environment).

-u *user* — Status is requested on all requests for the user who executed the command *rlpstat* on the specified printer (see the **–d** option).

id — Status is requested on the specified request IDs (as returned by *lp*(1)). All the request IDs must be for the same printer.

In the HP Clustered environment, all printers are attached to the cluster server and all spooling is handled as if the cluster were a single system. Remote spooling applies to spooling from or to machines outside of the cluster.

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

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. *Rmt* is normally started up with an *rexec*(3C) or *rcmd(3C)* call.

The *rmt* program accepts requests specific to the manipulation of magnetic tapes, performs the commands, then responds with a status indication. 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 *intro*(2) and *error-message* is the corresponding error string as printed from a call to *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*(2). 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*(2) operation using the specified parameters. The response value is that returned from the *lseek* call.

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 the *write*(2) call.

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*(2) and responds with **A***count-read*\n if the read was successful. Otherwise an error in the standard format is returned. If the read was successful, the data read is then sent.

I *operation count* Perform a MTIOCOP *ioctl*(2) command using the specified parameters. The 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).

Any other command causes *rmt* to exit.

DIAGNOSTICS

All responses are of the form described above.

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

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)) 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 &
```

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.*

BUGS

Normally it is not a good idea to restart *runacct* in the **SETUP** *state*. Run **SETUP** manually and restart via:

runacct *mmdd* **WTMPFIX**

If *runacct* failed in the **PROCESS** *state*, remove the last **ptacct** file because it will not be complete.

STANDARDS CONFORMANCE

runacct: SVID2

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 entries 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
```

will 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
```

will report 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

sam – system administration manager

SYNOPSIS

sam

DESCRIPTION

The command *sam* starts a menu-driven, screen-oriented user interface program that enables the user to perform system administration tasks without having or acquiring specialized knowledge of HP-UX. It is a self-guided tool and context sensitive help is available at any point. Information entered by the user is checked for validity as each field in a form is filled in. A status message is displayed whenever a task is performed.

Sam provides access to the following functions:

- User and Group Accounts Management
- File System Management
- Kernel and Device Configuration
- Line Printer Spooling Management
- UUCP Management
- Diskless Cluster Management
- Network Nodal Configuration
- Trusted Systems Management

As the Network Nodal Configuration and Trusted Systems features are not present on all system configurations, these areas are not always accessible through sam .

DEPENDENCIES

Requires an HP compatible terminal with programmable function keys and on-screen display of function key labels.

FILES

/usr/bin/sam - startup script for SAM

SEE ALSO

System Administration Tasks manual

NAME

savecore – save a core dump of the operating system

SYNOPSIS

/etc/savecore [**–nvcp**] [**–d** *dumpsystem*] *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 reads a number from the file *dirname*/**minfree**. The core dump is not done if the number of free 512-byte blocks on the file system that contains *dirname* is less than the number obtained from the **minfree** file. If **minfree** does not exist, *savecore* always writes out the core file. Note that repeated system crashes can result in multiple core files that consume large quantities of disk space, especially on machines with large physical memories.

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 file system (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.

–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 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*.

RETURN VALUE

Savecore returns the following exit status 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*.

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

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 the Hewlett-Packard Company.

SEE ALSO

sdf(4), du(1), df(1M).

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.

The following flags 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.

–s 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. You may specify up to five **d**'s. Using more than two, however, can result in an overwhelming amount of output.

Sdffsck also recognizes, and 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 *stderr*. Information generated because of the **–d** option and normal output is written to *stdout*; both are unbuffered.

Inconsistencies checked include:

1. Blocks claimed by more than one inode, or by 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. Blocks not accounted for anywhere;
5. Bad inode format;
6. 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 will also happen 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 you with the ability to perform the following functions on the specified *SDFdevice*:

1. Find the inode number of a file, given its full path name.
2. Examine and modify the contents of the superblock (volume header).
3. Examine and modify the contents of any inode or other file attribute.

Integer input to *sdffsdb* may 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 enter 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 you 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, then the record can be modified. If a right curly bracket "}" follows the number, then 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 you 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 you 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 Hewlett-Packard Company.

SEE ALSO

sdf(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*(1M) and *umount* (see *mount*(1M)) commands. *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 file system's "special file" (for example, **/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

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*(1M) 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 **–g** | **–n** | group–name [privileges]

setprivgrp -f file

DESCRIPTION

Setprivgrp associates a group with a kernel capability. This allows subsetting of super-user like privileges for members of a particular group or groups. In the first form, the first argument to *setprivgrp* is a group name, **–g**, or **–n** specifying a particular group, all groups, or no groups, respectively. The optional second and subsequent arguments are symbolic names indicating kernel capabilities. In the second form the **–f** option is used to specify a file, typically **/etc/privgroup**, from which group capabilities are set. The group access privileges are changed to reflect the specified kernel capabilities.

RTPRIO	accesses the *rtprio*(2) system call for setting real-time priorities.
MLOCK	accesses the *plock*(2) system call for locking process text and data into memory, and the SHM_LOCK command used with the *shmctl*(2) system call.
CHOWN	accesses the *chown*(2) system call.
LOCKRDONLY	accesses the *lockf*(2) system call for setting locks on files open for reading only.
SETRUGID	uses the *setuid*(2) and *setgid*(2) system calls to change, respectively, the real user ID or real group ID of a process.

Specifying no access privileges removes any privileges that may currently be assigned. Note that capabilities set by this command are not additive. If you want to add a capability for a particular group, you must respecify all capabilities that were already set for that group, in addition to the new capability.

The file named with the **–f** option should contain one or more lines in the following format:

–g | **–n** | group-name [privileges]

In the HP Clustered environment, privilege groups are maintained separately for each member of the cluster. The CHOWN privilege from a cnode is determined by the privilege groups set up on the root server.

Only the superuser can use this command.

ERRORS

Setprivgrp returns 1 if caller is not super user, and 2 if there is not enough table space to hold a new privileged group assignment.

AUTHOR

Setprivgrp was developed by the Hewlett-Packard Company.

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).

NAME

shutdown – terminate all processing

SYNOPSIS

/etc/shutdown [**–h** | **–r**] [**–d** *device*] [**–f** *lif_file*] [*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. The procedure is designed to interact with the operator; that is, the person who invoked *shutdown*. *Shutdown* may instruct the operator to perform specific tasks or supply certain responses before execution can resume.

Shutdown goes through the following steps:

All file systems' super blocks are updated; see *sync*(1M). This must be done before rebooting the system to ensure file system integrity.

All users logged in on the system are notified to log out by a broadcast message. The operator can display his/her own message at this time. Otherwise, a standard warning message is displayed.

All currently executing processes are terminated except those essential to the system or associated with the shutdown procedure.

All file systems are unmounted.

The next step depends on which of the following options are selected:

–h	Shutdown the system and halt.
–r	Shutdown the system and reboot automatically.
–d *device*	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*	Reboot from the specified file. If the filename is the NULL string, the power-up search sequence is made for a system. Otherwise, the filename has to follow the lif filename convention. The **–f** option can only be used with the **–r** option.
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** or **–h** is specified, the system is placed in *run-level s*; see *init*(1M).

In the HP Clustered environment, executing *shutdown* on the root server of a cluster causes all cluster nodes to also shut down. Each cluster node is shut down by a *shutdown* command executing locally. The arguments for these local *shutdown* commands are copied from the root server's *shutdown* command, with the exception of the **–d** and **–f** options.

If neither **–h** nor **–r** is specified from the root server, the local *shutdown* commands use **–r** when executing. Note that prior to executing *shutdown* on each cluster node, the root server's *shutdown* command disables the remote boot daemon. Thus, non-root-server cluster nodes cannot actually reboot until the remote boot daemon is enabled on the root server.

Executing *shutdown* on a client node only affects that node.

Only the super-user can execute the *shutdown* command.

RETURN VALUE

The most commonly encountered error diagnostic is *device busy*. This happens when a

particular file system could not be unmounted; see *mount*(1M).

EXAMPLES

To immediately reboot the system and run HP-UX again: **shutdown −r 0**

To halt the system in 5 minutes: **shutdown −h 300**

To go to init *run−level s* in 10 minutes: **shutdown 600**

DEPENDENCIES

Series 800

The **−d** and **−f** options are not supported.

SEE ALSO

init(1M), killall(1M), mount(1M), rbootd(1M), reboot(1M), sync(1M).

NAME

swapon – enable additional device or file system for paging and swapping

SYNOPSIS

/etc/swapon –a
/etc/swapon name|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. The system begins by swapping and paging 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 devices and file systems of the same priority.

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. By using the first field (*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 in **/etc/checklist** must specify a block special file. The *directory* specified for each **swapfs** entry in **/etc/checklist** must specify a directory on the file system to be enabled.

The second form of *swapon* announces individual block devices or file systems to be used for paging and swapping. Block devices must have been setup at system configuration time. File system swap is dynamic and can be added on a running system, but not removed.

name must specify a block special file if a block device is to be enabled for swapping,

directory specifies which file system is to enabled for swapping.

The *min limit reserve* and *priority* parameters default to zero and only have meaning if the first parameter passed to swapon is a *directory.*

min indicates the number of file system blocks to take from the file system at the time that *swapon()* is called.

limit indicates the maximum number of file system blocks the swap system is allowed to take from the file system.

reserve indicates the number of file system blocks that are saved for file system use only.

priority indicates the order in which space is taken from the file systems used for swap. Space is taken from the lower priority systems first.

WARNINGS

There is no way to stop paging and swapping on a device or file system.

Exercise due caution when enabling swap space on a device or file system that may be unmounted during system operation or removed from the system.

The system will allocate no less than the amount specified in "min", however, to make the most efficient use of space, more than the amount requested might be taken from the file system. The actual amount taken will not exceed the number of file system blocks indicated in "reserve".

If more swap blocks are needed for one particular file system than those already added, additional calls can be made to swapon().

EXAMPLES

swapon /swap 10 3000 20 0

swapon /mnt 0 0 0 1

swapon /extra 0 2580 1000 2

FILES

/dev/dsk/c#d#s# Normal paging devices.

AUTHOR

Swapon was developed by the University of California, Berkeley.

SEE ALSO

swapon(2), checklist(4).

NAME

sync – synchronize file systems

SYNOPSIS

sync [**–l**]

DESCRIPTION

Sync executes the *sync*(2) system call. 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*(1M) to execute *sync* automatically at periodic intervals.

Options

–l Execute *lsync*(2) system call instead. 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*] [**–l**] [**–d** *directory* ...]

DESCRIPTION

Syncer is a program that periodically executes *sync*(2) or *lsync*(2) at an interval determined by the input argument *seconds*. 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.**

The **–l** option causes *syncer* to use *lsync*(2) instead of *sync*(2). The *lsync* system call will perform a local sync, while the *sync* system call will perform a cluster-wide sync (see *sync*(2) for details).

The **–d** option is used to open directories for cache benefit. All directories must be specified by their full path name. If the **–d** option is not used, no directories will be opened.

AUTHOR

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

SEE ALSO

brc(1M), init(1M), sync(1M), sync(2).

NAME

sysdiag – online diagnostic system interface

SYNOPSIS

sysdiag [filename]

DESCRIPTION

Sysdiag is the command interpreter for the online diagnostic system. 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 *Hardware Support Documentation Set*. *Sysdiag* accepts commands from either standard input or the specified *filename*.

The online diagnostic system allows the user to diagnose the computer system hardware without placing the system into single-user mode. Certain restrictions apply to running diagnostics in an online environment. These restrictions are necessary to protect user data. Each diagnostic program defines which operations destroy data and which do not. Typically, those operations that destroy data cannot be run in a multi-user environment. For further information, see the reference manual for each of the diagnostic programs.

Command Summary

abort	Abort an active diagnostic program.
ci or !	Fork and exec a shell.
exit	Exit *sysdiag*.
hardcopy	Produce a hardcopy of all *sysdiag* and diagnostic program input and output.
help or ?	Provide online help for *sysdiag* and diagnostic programs.
install	Add a diagnostic program to the diagnostic system.
list	List the installed diagnostic programs.
purge	Remove a diagnostic program from the diagnostic system.
redo	Edit and execute a previous command.
resume	Restart execution of a diagnostic program.
run	Begin execution of a diagnostic program.
showactive	Show active diagnostic programs.
suspend	Suspend execution of a diagnostic program.
use	Redirect standard input for *sysdiag*.
wait	Wait for background diagnostic programs to terminate.

AUTHOR

Sysdiag was developed by HP.

FILES

/usr/diag/bin/*	diagnostic programs
/usr/diag/install/*	installation information
/usr/diag/security	access list for diagnostic users
/usr/diag/cat/*	native language support catalogs
/dev/diag/*	diagnostic special files

SEE ALSO

Hardware Support Documentation Set, Volumes 4 & 5 sysdiag.1m . sysdiag.1m . sysdiag.1m . sysdiag.1m

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**. 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 <**syslog.h**>. *Syslogd* reads from an Internet domain socket specified in **/etc/services** or from the named pipe **/dev/log**.

Syslogd configures when it starts up and whenever it receives a hangup signal. Lines in the configuration file have 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* describing the part of the system that generated the message, a dot, and a *level* indicating the severity of the message. Symbolic names may be used. An asterisk selects all facilities. All messages of the specified level or higher (greater severity) are selected. More than one facility may 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" may be used to disable a particular facility. For example,

*.debug;mail.none

Sends all messages *except* mail messages to the selected file.

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.
- A hostname preceded by an at sign (@). Selected messages are forwarded to the *syslogd* on the named host.
- A comma separated list of users. Selected messages are written to those users if they are logged in.
- An asterisk. Selected messages are written to all logged-in users.

Blank lines and lines beginning with '#' are ignored.

For example, the configuration file:

kern,mark.debug	/dev/console
*.notice;mail.info	/usr/spool/adm/syslog
*.crit	/usr/adm/critical
kern.err	@ucbarpa
*.emerg	*
*.alert	eric,kridle
*.alert;auth.warning	ralph

logs all kernel messages and 20 minute marks onto the system console, all notice (or higher) level messages and all mail system messages except debug messages into the file **/usr/spool/adm/syslog**, and all critical messages into **/usr/adm/critical**. Kernel messages of error severity or higher are forwarded to the machine **ucbarpa**. In this example, all users are informed of any emergency messages, the users "eric" and "kridle" are informed of any alert messages, and the user "ralph" is informed of any alert message or any warning message (or higher) from the authorization system.

The options are:

−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 bring *syslogd* down, it should be sent a terminate signal (e.g., kill `cat /etc/syslog.pid`).

AUTHOR

Syslogd was developed by the University of California, Berkeley.

FILES

/etc/syslog.conf	the configuration file
/etc/syslog.pid	the process ID
/dev/log	the named pipe *syslog* reads from

SEE ALSO

logger(1), syslog(3C).

NAME

sysrm – remove optional HP-UX products (filesets)

SYNOPSIS

/etc/sysrm [**–S** *series*] *fileset* ...

DESCRIPTION

Sysrm removes all files associated with an optional product (fileset) from an HP-UX file system. This is usually done to recover mass storage space. Only the superuser can execute *sysrm*.

A *fileset* argument is the name of a file in the filesets directory that contains a list of files (path names) associated with the product being removed. *Sysrm* processes these path names relative to the current working directory. Normally, each line in a fileset file is an absolute path name, so the present working directory is irrelevant.

Sysrm does not remove files listed in the "noremove" file (see FILES below). Also, it does not remove any directories, nor any files in filesets required for a minimum system:

KERN_BLD
UX_CORE
TOOL
C_MIN
PROG_MIN

After removing all removable files in a fileset, *sysrm* removes the fileset file itself.

Diskless Features

In clustered systems, the **–S** option is available:

–S *series* Specify Series 300 (**–S300**) or Series 800 (**–S800**) to remove the *fileset* for an architecture other than the one on which *sysrm* is run. This option is useful on the root server of a mixed-architecture diskless cluster.

On a mixed-architecture cluster server, the filesets directory is a context dependent file (CDF). If a *fileset* is present for more than one architecture type (Series), only the files specific to that architecture's fileset are removed. Files shared between architectures, and CDF elements which appear in other architectures' versions of the fileset file, are not removed.

For CDF elements, list in the "noremove" file the explicit path to the element, not the name of the whole CDF. Otherwise *sysrm* uses its process context, not the **-S** option, to distinguish the elements.

When *sysrm* removes the last element of a CDF, it also removes the CDF.

RETURN VALUE

Sysrm returns 0 if it attempts to read any filesets and remove any files, whether or not it succeeds. It returns 1 if it detects any invocation errors, such as being run with no arguments or by other than the superuser.

DIAGNOSTICS

Sysrm prints to standard output each file name to be removed before removing it. It prints to standard error a warning message in each case where it cannot open a specified *fileset* file.

EXAMPLES

Remove the NAMERICA and PC filesets:

sysrm NAMERICA PC

WARNINGS

The system should be in single-user mode when *sysrm* is executed. Ensure that only *init*, *sh*, and other required processes are running.

AUTHOR

Sysrm was developed by HP.

FILES

/etc/filesets	directory where fileset lists are kept
/etc/filesets/noremove	optional file containing file names that must not be removed from the system

SEE ALSO

update(1M).

NAME

tic – terminfo compiler

SYNOPSIS

tic [−**v**[*n*]] file ...

DESCRIPTION

Tic translates terminfo files from the source format into the compiled format. The 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 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 there instead of **/usr/lib/terminfo**.

Some 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**, it should check for an existing compiled entry.

STANDARDS CONFORMANCE

tic: SVID2

NAME

tunefs – tune up an existing file system

SYNOPSIS

/etc/tunefs *tuneup-options special*

DESCRIPTION

Tunefs is designed to change a file system's dynamic parameters that affect the layout policies. The parameters that are to be changed are indicated by the flags given below:

–a *maxcontig* This specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay (see **–d** below). The default value is one, since most device drivers require one interrupt per disk transfer. Device drivers that can chain several buffers together in a single transfer should set this to the maximum chain length.

–d *rotdelay* This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

–e *maxbpg* This indicates the maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. Typically this value is set to about one quarter 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 big 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* This value specifies the percentage of space held back from normal users, i.e., the minimum free space threshold. The default value used is 10%. This value can be set to zero; if it is, up to a factor of three in throughput will be lost over the performance obtained at a 10% threshold. Note that if the value is raised above the current usage level, users will be unable to allocate files until enough files have been deleted to get under the higher threshold.

–A This option specifies that redundant super-blocks, as well as the super-block, are to be modified as indicated above.

special This is the name of the file system that will be tuned. It is either a block or character special file for an unmounted volume or volume section.

WARNINGS

This program should work on mounted and active file systems. Because the super-block is not kept in the buffer cache, the program will take effect only if it is run on dismounted file systems. If run on the root file system, the system must be rebooted.

You can tune a file system, but you can't tune a fish.

AUTHOR

Tunefs was developed by the University of California, Berkeley.

SEE ALSO

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 that path 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) option, however, the user can specify the terminal type used.

With the *file* option 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

/etc/update [**−cm**] [**−s** *source*] [**−S** *series*] [**−P** *port*] [**−d** *destination*] [**−r** *kernel_gen_file*] [**−b** *partition*] [**−f** *file*] [*fileset...*]

/etc/updist [**−cm**] [**−s** *source*] [**−S** *series*] [**−P** *port*] [**−d** *destination*] [**−f** *file*] [*fileset...*]

DESCRIPTION

Update is used interactively or non-interactively, and from local media or a remote "netdist" server system, 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.

Updist is similar to *update*, but 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 network server daemon, *netdistd*(1M), finds the files in this special directory and supplies them to a remote *update* process at its request.

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. 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 media consist of simple *tar*(1) archives containing product files and directories 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. If an attempt is made to load multiple media non-interactively, *update* refuses to begin loading; *updist* loads the first media unit only, then terminates with a warning message.

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 superuser can run *update* and *updist*.

Options

Update and *updist* support the following options when used non-interactively:

−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 number, and any associated fileset flags (see *update*(4)). The output is usable as input to the **−f** option.

−m (match) Load new versions of all filesets currently installed on the system. The **filesets** directory is used to determine the currently loaded set. If fileset names change between HP-UX releases, *update* and *updist* use internal tables to correctly convert old names to corresponding names in the new release.

−s *source* Specify a non-default source for the update. *Source* can be:

- Hostname of a netdist server system running the *netdistd*(1M) daemon
- Local block-special file representing a nine-track tape
- Local character-special file representing a cartridge tape

The default *source* is **/dev/update.src** on Series 300 systems and **/dev/rmt/0m** on Series 800 systems.

−S *series* Specify Series 300 (**−S300**) 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 from old S800 media that don't contain a type designation, and if necessary when loading files from a netdist server system running the *netdistd*(1M) daemon.

−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 file (see FILES below).

−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 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 inspects or modifies the kernel, since *update* must inspect or alter the generation file to succeed. On Series 300 systems, *kernel_gen_file* is typically **/etc/conf/dfile**. On Series 800 systems, it is typically **/etc/conf/gen/S800**. There is no default value.

−b *partition* (*Update* only; S800 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. It 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 **#**.

The **−m** and **−f** options, and naming filesets explicitly, are mutually exclusive.

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 a user 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. Likewise, interactive unselection of a fileset causes automatic unselection of its other required filesets, unless the user explicitly selected them or selected other filesets that require them.

Update creates or rewrites fileset lists in 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 *Netdistd* Server Master Files

Updist transfers filesets from factory-supplied update media to a special tree under the local file system of an *update* server system. *Updist* prepares the filesets for use by the *netdistd*(1M) server process. *Updist* differs from *update* in these ways:

- The default destination is **/netdist**.
- Fileset files are not created.
- Customize scripts are not run.
- Other 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 (disks must be listed in **/etc/mnttab**).

Update and *updist* break hard links caused by re-creating updated files. They also follow symbolic links and update the targets of the symbolic links. *Update* records in fileset files the symbolic path to updated files, not the actual path; even when traversing a symbolic link.

HP Clustered Environment

Update and *updist* recognize filesets as containing Series 300 or Series 800 files. When loading files for one Series onto a file system running on the other Series (that is, onto the cluster server of a mixed cluster), *update* creates context-dependent file (CDF) elements as required (specified by the media). Also, the fileset and system directories on a mixed cluster (see FILES below) should already be CDFs. *Updist* maintains separate subdirectories for each Series.

Update does not install cluster servers or convert standalone systems to cluster servers. Use *sam*(1M) for that purpose.

RETURN VALUE

Interactive invocation always returns 0. Command line invocation returns:

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 displayed during interactive execution, and error messages resulting from invalid non-interactive invocation, are self-explanatory. 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 "NAMERICA" and "PC" on the system using the default source device and destination directory:

update NAMERICA PC

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), rebooting when done if necessary, and reading the current kernel configuration from **/etc/conf/dfile.old**:

update −mr −g /etc/conf/dfile.old

Extract fileset "TEXT" 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

WARNINGS

Always review the log file after an update for relevant warnings and messages.

Update can reboot the system as part of an update. Flags associated with each fileset indicate the necessity for a reboot. In interactive mode, *update* warns the user if such a fileset is selected. If the system is not in a quiescent state, the user can proceed, or quit *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, or for delayed execution of customize scripts, *update* reboots the system twice. The first reboot brings up the new kernel and allows customize scripts to run. The second reboot restarts the system in its normal state after all customization is complete.

If both rebooting and non-rebooting filesets are loaded, *update* leaves a file named **/customize** on the system. This is the master customize script that invokes all others and that 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 *sysrm*(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 *sysrm* 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.

Sysrm does not know how to remove packages loaded by *updist*. To remove *updist*-package files, run **rm −r** on the package directories under the netdist source tree, and 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.

Run interactively, *update* and *updist* depend on the TERM environment variable to tell them 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.

DEPENDENCIES

Networking

Update and *updist* refuse to update files on remote systems over NFS or RFA network connections. They pre-mount 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, and give warnings about remote files not loaded.

AUTHOR

Update and *updist* were developed by HP.

FILES

/dev/update.src	default source for Series 300 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 file lists are stored
/system	directory containing important information and customization scripts for each fileset
/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 reboot is required
/etc/services	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

sam(1), tar(1), sysrm(1M), mount(1M), netdistd(1M), update.6.5(1M), update(4), *HP-UX System Administrator Manual*.

NAME

update.6.5 – update optional HP-UX products (Series 300 6.5 version)

SYNOPSIS

/etc/update.6.5 [**–nm**]

DESCRIPTION

Update.6.5 is used to process a periodic update of an HP-UX optional product. It is an old version which recognizes *cpio*-format update media. It will continue to be supported temporarily.

The 7.0 and later versions of *update* recognize *tar*-format update media and are backward compatible with older Series 800 *tar*-format media.

Only the super-user can execute *update.6.5*.

The update process is interactive. It prints information about addresses of mass storage devices to be used, and gives the user choices to change them. After the proper addresses are set and the user has selected the choice to read the table of contents on the update source media, *update.6.5* prints loading options for the products distributed on the update source media. The user then must select which optional products are to be loaded. There is also a choice to load all optional products which can be used to cut down on time delays for user interaction.

When all desired products are loaded, the **exit** choice is used to terminate the update process and reboot the system.

See the 6.5 Install and Update Manual supplied with your system for an in-depth explanation of how to use *update.6.5*.

Options

The options are:

–n No reboot. In conjunction with pre-7.0 media designed for loading without rebooting, this option suppresses system reboots. It allows installation and update of software without altering the current system state or disturbing system users. The media must also be marked to allow loading without rebooting.

–m Interact with the display without using escape codes, cursor controls, and other display manipulations. This option is provided for use with unsupported terminals.

WARNINGS

Update.6.5 reboots the system when it is first called. Be sure you have a recent backup of the system before performing an update.

There is no way to prevent accidentally updating an older version of a product over a newer one.

Hewlett-Packard Company supports only those terminals in the *terminfo* data base that are included in the current list of supported devices for the HP-UX release being used. Other terminal model entries may be included in the *terminfo* data base that are not officially supported. If you choose to use such devices, they may or may not work correctly.

FILES

/etc/filesets	directory where partition file lists are kept
/system	directory which contains important information and customization scripts for each fileset
/tmp/update.log	log file containing information about success/failure of the update process

SEE ALSO

cpio(1), tar(1), sysrm(1M), update(1M)

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 *uucp*(1) system required files and directories. Within the *uucp* makefile, it is executed before the installation occurs. *Uucheck* also checks for some obvious errors in the **/usr/lib/uucp/Permissions** file.

Options

–v Print a detailed explanation of how *uucp*(1) 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*(1).

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

uucico(1M), uusched(1M), uucp(1), uustat(1), uux(1).

UUCP, a tutorial in *HP-UX Concepts and Tutorials.*

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 the **/usr/lib/uucp/Systems** file. *Uucico* then executes all requests for work and logs the results.

Options

The options are as follows:

–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 the 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 – 9. More debugging information is given for larger values of *debug_level*.

Uucico is usually started by a local program (e.g., *cron*(1M), *uucp*(1), or *uuxqt*(1M)). It should *only* be directly initiated by a user when debugging.

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.

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

cron(1M), uusched(1M), uutry(1M), uucp(1), uustat(1), uux(1).

UUCP, a tutorial in *HP-UX Concepts and Tutorials.*

NAME

uuclean – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [*options*]

DESCRIPTION

Uuclean will scan the spool directories for files with the specified prefix and delete all those which are older than the specified number of hours.

The following options are available.

–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 may 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 will be deleted if the prefix test is satisfied. (default time is 72 hours)

–w*file* The default action for *uuclean* is to remove files which are older than a specified time (see **–n** option). The **–w** option is used to find those 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 will go to the standard output.

–s*sys* Only files destined for system *sys* are examined. Up to 10 **–s** arguments may be specified.

–m*file* The **–m** option sends mail to the owner of the file when it is deleted. If a *file* is specified then an entry is placed in *file*.

This program is typically started by *cron*(1M).

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*(1). See *uucleanup*(1M) for an alternative interface that works with "HoneyDanBer style" *uucp*(1).

FILES

/usr/lib/uucp	directory with commands used by *uuclean* internally
/usr/spool/uucp/*	spool directory

SEE ALSO

cron(1M), uucp(1), uux(1), uucleanup(1M).

UUCP, a tutorial in *HP-UX Concepts and Tutorials.*

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

The following options are available:

−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 may 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*

−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*(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 *HP-UX Concepts and Tutorials.*

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*(1M), except that *uugetty* supports using the line in both directions. This allows users to login, but, if the line is free, *uucico*(1), *cu*(1), and *ct*(1) can dial out. When devices are used with *uucico*(1), *cu*(1), and *ct*(1) lock files are created. Therefore, when the call to *open*(2) returns (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*(1), *cu*(1), *ct*(1) or someone trying to login. See *getty*(1M) for more information.

Note that with the option **–r**, several <carriage-return> characters might be required before the login message is output. When *uucico* is trying to login, 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 the ... is 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 will 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

uucico(1M), getty(1M), init(1M), tty(7), ct(1), cu(1), login(1), ioctl(2), gettydefs(4), inittab(4).

UUCP tutorial in *HP-UX Concepts and Tutorials.*

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 **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 columnated so there may 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 which 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 don't 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 which 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 gotten 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.

DEPENDENCIES

Series 300

The uucp spool files are found in the **/usr/spool/uucp** directory, instead of in subdirectories of that directory.

AUTHOR

Uuls was developed by the Hewlett-Packard Company.

SEE ALSO

mail(1), uucp(1), uuto(1), uux(1), uuxqt(1M), stat(2).

UUCP, a tutorial in *HP-UX Concepts and Tutorials.*

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)).

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 un-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 "*" prepended, 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*(1M) from the following entry in **/usr/spool/cron/crontab**:

```
39 * * * * /bin/su uucp -c */usr/lib/uucp/uudemon.hour > /dev/null*
```

Options

The two options are for debugging purposes only.

–x *debug_level*	Output debugging messages
–u *debug_level*	Pass as **–x** to *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 *HP-UX Concepts and Tutorials.*

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 *HP-UX Concepts and Tutorials.*

NAME

uusnaps – sort and embellish uusnap output

SYNOPSIS

uusnaps

DESCRIPTION

Uusnaps runs *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*(1) to check.

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, a tutorial in *HP-UX Concepts and Tutorials.*

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. The following options are available:

–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*(1M) 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, a tutorial in *HP-UX Concepts and Tutorials.*

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 the use of the *uux*(1) command. *Uux*(1) generates **X.** files and places them into the spool directory, where *uuxqt* searches for them. For each **X.** file, *uuxqt* determines if 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. *Uux* then performs the 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.

The **–x** *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.

FILES

/usr/lib/uucp/Permissions
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/uucp/LCK*

SEE ALSO

uucico(1M), uucp(1), uustat(1), uux(1).

UUCP, a tutorial in *HP-UX Concepts and Tutorials.*

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. Five files (**conf.c**, **config.h**, **devices**, **libs_file**, **Makefile**) are created by *uxgen*. The file **devices** contains a list of I/O devices, pseudo drivers and major numbers assigned to them. This file is used by the commands *insf*(1M), *mksf*(1M) and *lssf*(1M) for making and listing special files. The file **libs_file** contains a macro recognized by *make*(1) specifying the libraries needed to produce a kernel with all the subsystems and file systems included in *infile*. 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*(1) macro defined in **libs_file**. The concatenation of the files **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. The file **hp-ux** can then be booted. See the *HP-UX System Administrator Manual* for information on how to include or remove a subsystem or a 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 to build the kernel. By convention, *infile* is placed in the directory named **gen** (usually, **/etc/conf/gen**).

The procedure for building a kernel follows:

Instruction	Command
1. Switch current directory to **gen**.	cd **/etc/conf/gen**.
2. Edit or create *infile*.	
3. Execute *uxgen*, thus making the new kernel, named *../infile*/**hp-ux**.	**uxgen** *infile*
4. Save the old **/hp-ux** and **/etc/devices** files.	**cp /hp-ux /SYSBCKUP** **cp /etc/devices /etc/DEVBCKUP**
5. Move the new **hp-ux** and **devices** file into place.	**mv ../***infile***/hp-ux /hp-ux** **mv ../***infile***/devices /etc/devices**
6. Follow reboot instructions.	See *shutdown*(1M).

It is beyond the scope of this manual page to give a complete description of the statements that can be used in *infile*. A "loose" syntax description for most statements is given below. See the *HP-UX System Administrator Manual* for more information.

The statements used in *infile* form a simple C-like language. Before being read, *infile* is passed through the C preprocessor. This allows features such as comments, macros, file inclusion and conditional statements. See *cpp*(1) for more information about the C preprocessor.

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. The* **/etc/master** file contains "subsystem", "file system", "driver", "pseudo driver", "streams module" and "tunable definition" statements which describe kernel software subsystems, file systems, I/O drivers, pseudo drivers, streams modules, tunable parameters and major number assignments for software supplied by HP. Only users who write kernel software (such as subsystems, file systems, drivers, pseudo drivers) need to understand these statements. They are described in the *HP-UX System Administrator Manual* . All other statements are briefly described below:

args on *module-ID* **lu** *integer* **section** *integer* **;**
Obsolete. The first disk section specified in the **swap on** statement is used for writing the argument list when calling *exec*(2).

console on *module-ID*
Specify the module name used for the system console.

dumps on *module-ID* **lu** *integer* **section** *integer* **;**
Specify the module name, logical unit number and section number used for writing the operating system image after the operating system detects a fatal error (panic).

include *identifier* **;** Include a pseudo driver, subsystem, file system or streams module in the kernel.

io {
identifier **lu** *integer* **address** *integer* **;**
. . .
identifier **address** *integer* {
identifier **lu** *integer* **address** *integer* **;**
}
. . .
}
Specify the number of I/O devices and how they are connected.

remove *identifier* **;** Remove a pseudo driver, subsystem, streams module or file system that was previously included with an include statement.

root on *module-ID* **lu** *integer* **section** *integer*
[mirrored on *module-ID* **lu** *integer* **section** *integer* **];**
root on remote ;
Specify the module name, logical unit number and section number for the root of the file system ("/"). The optional portion specifies the root mirror. If a root mirror is specified, the module-ID must be "disc2" for both the mirror disk and the disk containing the root of the file system. Disk mirroring also requires that the section on the mirror disk and the root section be identical (see *mirror*(1M)). For a diskless system, the root of the file system can be specified as remote.

swap on *module-ID* **lu** *integer* **section** *integer*
[mirrored on *module-ID* **lu** *integer* **section** *integer* **]**
[*module-ID* **lu** *integer* **section** *integer* **] ... ;**
swap on remote ;

Specify the module name, logical unit number and section number used for swapping. Multiple swap areas can be defined. Only the first swap area listed may be mirrored. If a root mirror is specified, the module-ID must be "disc2" for both the mirror disk and the disk containing the primary swap space. Disk mirroring also requires that the section on the mirror disk and the primary swap section be identical (see *mirror*(1M)). For a diskless system, the swap area can be specified as remote.

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 with a "tunable definition" statement in the **/etc/master** file.

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**).

bufpages The number of memory pages allocated to the file-system buffer cache. Each page is 2048 bytes long. 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 a diskless cnode (specified in seconds).

dmmax
dmmin
dmshm
dmtext The values of these parameters are used as described below in SWAP SPACE PARAMETER INTERACTION.

dskless_cbufs The number of pages allocated to the cbuf (cluster buffer) pool. Each page is 2048 bytes long.

dskless_mbufs The number of pages allocated to the mbuf pool. Each page is 2048 bytes long.

dskless_node A flag that identifies a diskless node or root server. A value of **1** identifies a diskless node. A value of **0** identifies the root 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 USA daylight saving time should be used.

iomemsize The amount of memory available to the I/O system via the kernel function **io_get_mem()**. The value is in bytes.

itebuflines Obsolete (see **scroll_lines**).

maxdsiz The maximum size (in pages) of a process's data segment.

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 * 1 kbyte 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 real-time priority by which the process **/etc/netisr** is scheduled, and by which all networking runs. Otherwise, **netisr_priority** is ignored.

netmeminit If a networking subsystem is configured, **netmeminit** specifies the number of bytes of memory to be preallocated at system initialization time. Otherwise, **netmeminit** is ignored.

netmemmax If a networking subsystem is configured, **netmemmax** specifies the maximum number of bytes of memory to be used for networking. Otherwise, **netmemmax** is ignored.

netmemthresh If a networking subsystem is configured, a non-negative **netmemthresh** specifies the number of bytes at which the networking subsystem begins to overreserve memory. If **netmemthresh** is **−1**, overreservation is turned off for the value of **netmemmax** supplied. Other negative values for **netmemthresh** are reserved and should not be used. If no networking subsystem is configured, **netmemthresh** 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.

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 x NUM_CNODES";).

retry_alive_period The period to continue checking for alive messages from a diskless 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 selftest (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 diskless 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 D of the *HP-UX System Administrator Manual* contains a table which lists the values these parameters must have for given values of **maxdsiz**, **maxssiz**, **maxtsiz** or **shmmax**.

EXAMPLES

```
include             mirror;
include             nfs;
include             nsdiag0;

console on          mux0;
root on             disc2 lu 0 section 0
  mirrored on       disc2 lu 1 section 0;
dumps on            disc0 lu 0 section 1;
swap on             disc0 lu 0 section 1
                    disc0 lu 1 section 1;

acctresume          4;
acctsuspend         2;
bufpages            0;
check_alive_period  4;
dmmax               2048;
dmmin               32;
dmshm               2048;
dmtext              2048;
dskless_cbufs       "(DSKLESS_MBUFS*2 + 1)";
dskless_mbufs       "(((SERVING_NODE*NUM_CNODES)/4) + 1)";
dskless_node        1;
dst                 1;
iomemsize           "(4 * NUM_IOTREE_RECS * NBPG)";
maxdsiz             0x8000;
maxssiz             0x1000;
maxswapchunks       512;
maxtsiz             0x8000;
maxuprc             25;
maxusers            32;
minswapchunks       1;
msgmap              100;
msgmax              8192;
msgmnb              16384;
msgmni              50;
msgseg              1024;
msgssz              8;
msgtql              40;
nbuf                0;
ncallout            "(16+NPROC+USING_ARRAY_SIZE+SERVING_ARRAY_SIZE)";
netisr_priority     100;
netmeminit          0;
netmemmax           "(512 * NETCLBYTES)";
netmemthresh        0;
nfile               "(16*(NPROC+16+MAXUSERS)/10+32+2*NETSLOP)";
nflocks             200;
ngcsp               "(8 * NUM_CNODES)";
```

```
ninode              "(NPROC+16+MAXUSERS+32+2*NPTY+SERVER_NODE*18*NUM_CNODES)";
nproc               "(20 + 8 * MAXUSERS)";
npty                60;
nstlbe              0;
ntext               "(24 + MAXUSERS + NETSLOP)";
num_cnodes          0;
retry_alive_period  21;
scroll_lines        100;
selftest_period     120;
semaem              16384;
semmap              10;
semmni              10;
semmns              60;
semmnu              30;
semume              10;
semvmx              32767;
server_node         0;
serving_array_size  "(MAXUSERS*(SERVER_NODE*NUM_CNODES + 2))";
shmmax              0x4000000;
shmmni              100;
shmseg              12;
timeslice           "(HZ/10)";
timezone            420;
unlockable_mem      0;
using_array_size    "(NPROC)";

io {
        cio_ca0 address 28 {
                hpib0 address 0 {
                        disc0 lu 0 address 0;
                        disc0 lu 1 address 1;
                        disc0 lu 2 address 2;
                }
                mux0 lu 0 address 1;
                hpib0 address 2 {
                        lpr0 lu 0 address 0;
                        tape0 lu 0 address 2;
                        instr0 lu 0 address 7;
                }
                hpfl0 address 3 {
                        disc2 lu 0 address 0;
                        disc2 lu 1 address 1;
                }
        }
}
```

AUTHOR

Uxgen was developed by HP.

FILES

/etc/devices
/etc/master

SEE ALSO

make(1), insf(1M), lssf(1M), mksf(1M), shutdown(1M), devices(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 will be used unless the environment variable EDITOR indicates an alternate editor. *Vipw* performs a number of consistency checks on the password entry for *root*, and will 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

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* will spawn 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 of the 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 then 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* will look for requests on. If no lan devices are specified then the default lan device is used. The major number for this device must correspond to a CIO 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* will create a portal named *calldev* in **/dev.** Whenever this device is opened, *vtdaemon* will will spawn a server that will create a connection to the system specified by *nodename* via the lan device specified. If no lan device is specified then 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 will remove 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* reference in *HP-UX Concepts and Tutorials: Shells and Miscellaneous Tools.*

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

When *wall* is invoked without arguments, the standard input is read 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 the *file* option 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 be super-user to override any protections the users may have invoked (see *mesg*(1)).

Wall has timing delays, and will take at least 30 seconds to complete.

DIAGNOSTICS

"Cannot send to ..." when the open on a user's tty file fails.

AUTHOR

Wall was developed by AT&T and HP.

FILES

/dev/tty*

SEE ALSO

mesg(1), write(1).

EXTERNAL INFLUENCES

International Code Set Support

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

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*(1) and *ps*(1) commands.

FILES

/etc/passwd

SEE ALSO

ps(1), who(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)).

STANDARDS CONFORMANCE

whodo: SVID2

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 the 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 and Series 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

a.out_300(4), a.out_800(4), ar(4), magic(4).

NAME

a.out – assembler and link editor output

Remarks:

This manual page describes the *a.out* file format for Series 300 computers. Refer to other *a.out*(4) manual pages for descriptions of other valid implementations.

DESCRIPTION

A.out is the output file of the link editor *ld*. *Ld* will make **a.out** executable if there were no linking errors and no unresolved external references. The assembler *as* produces non-executable files with the same structure.

File *a.out* has seven sections: a header, the program text and data segments, a pascal interface section, a symbol table, information for debugger support, and text and data relocation information (in that order). The pascal interface text will only be present in those pascal code segments that have not been linked. The last three sections may be missing if the program was linked with the **–s** option of *ld*(1) or if the symbol table, debug information, and relocation bits were removed by *strip*(1). Also note that if there were no unresolved external references after linking, the relocation information will be removed.

The file section containing information for debugger support has three tables – the debug name table (DNTT), the source line table (SLT), and the value table (VT). These tables contain symbolic information used by the HP-UX debugger *cdb*(1). 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*(2) system call.

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 stamp */
        short   a_unused;
        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;       /* size of text relocation info */
        long    a_drsize;       /* size of data relocation info */
        long    a_pasint;       /* size of interface text */
        long    a_lesyms;       /* size of symbol table */
        long    a_dnttsize;     /* debug name table size */
        long    a_entry;        /* entry point of program */
        long    a_sltsize;      /* source-line table size */
        long    a_vtsize;       /* value table size */
        long    a_spare3;
        long    a_spare4;
};
```

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;
        short           n_unused;
};
```

Following this structure is *n_length* ascii characters which compose the symbol name.

The *n_type* field indicates the type of the symbol; the following values are possible:

00 undefined symbol
01 absolute symbol
02 text segment symbol
03 data segment symbol
04 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.

The start of the symbol table is given by the macro LESYM_OFFSET(hdr).

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:

00 indicates the reference is to the text segment;
01 indicates the reference is to initialized data;
02 indicates the reference is to bss (uninitialized data);
03 indicates the reference is to an undefined external symbol.

The field *r_symbolnum* 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 field *r_length* indicates the length of the datum to be relocated.

00 indicates it is a byte
01 indicates it is a short
02 indicates it is a long
03 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).

SEE ALSO

as_300(1), as_800(1), ld(1), nm_300(1), nm_800(1), strip(1), a.out_800(4), magic(4).

NAME

a.out – assembler and link editor output

SYNOPSIS

#include <a.out.h>

Remarks:

This manual page describes the *a.out* file format for Series 800 computers. Refer to other *a.out*(4) manual pages for descriptions of other valid implementations.

DESCRIPTION

The file name **a.out** is the output file from the assembler *as*(1), the compilers, and the linker *ld*(1). The assembler and compilers create relocatable object files ready for input to the linker; the linker creates executable object 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.

Programs for the Series 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, and all data in the second quadrant of the $PRIVATE$ space. (Thus, it is sufficient to use a 32-bit pointer to access any address in either 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:

$MILLICODE$	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
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 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.

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;                /* 0x020b */
    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

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 */
};
```

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 at 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 pt.*/
#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 the Series 800, each fixup request is a five-word structure that describes 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_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_STATEMENT Define the beginning of 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_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_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_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), a.out_300(4).

NAME

acct – per-process accounting file format

SYNOPSIS

#include <sys/acct.h>

DESCRIPTION

Files produced as a result of calling *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*(2) and turned off by an *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).

BUGS

The *ac_mem* value for a short-lived command gives little information about the actual size of the command, because *ac_mem* may be incremented while a different command (e.g., the shell) is being executed by the process.

STANDARDS CONFORMANCE

acct: SVID2, XPG2

NAME

ar – common archive file format

SYNOPSIS

#include <ar.h>

DESCRIPTION

Ar(1) is used to concatenate several files into an archival file. Archives are used mainly as libraries to be searched by the link editor *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 *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  ARFMAG   "`\n"        /* header trailer string */

struct  ar_hdr {               /* file member header */
   char    ar_name[16];        /* '/' terminated file member name */
   char    ar_date[12];        /* file member date */
   char    ar_uid[6];          /* file member user identification */
   char    ar_gid[6];          /* file member group identification */
   char    ar_mode[8];         /* file member mode (octal) */
   char    ar_size[10];        /* file member size */
   char    ar_fmag[2];         /* header trailer string */
};
```

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*(1) is used. Note that older versions or *ar*(1) 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 newline 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. Further details can be found in the *a.out*(4) manual page for each machine.

SEE ALSO

ar(1), ld(1), strip(1), a.out(4), magic(4).

CAVEATS

Strip(1) will remove all archive symbol entries from the header. The archive symbol entries

must be restored via the **ts** option of the *ar*(1) command before the archive can be used with the link editor *ld*(1).

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*(2). Access to the auditing system is restricted to superusers.

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*(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

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" will 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
```

will refer to the appropriate files.

A directory is changed to a hidden directory using *chmod*(1). In the above example, if **/etc/inittab** were an ordinary directory,

chmod +H /etc/inittab

would be used to make it a hidden directory. Invoking:

chmod −H /etc/inittab+

would make **/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 may 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 HP9000 Series 300 workstations some of the workstations might have HP98248A 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 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(2), chmod(1), context(5), find(1), getcdf(3C), getcontext(2), getcontext(1), ls(1), pwd(1), makecdf(1M), showcdf(1), tar(1).

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*/
    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, 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*/
/*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.

The *cdf_loc_lsb* and *cdf_loc_msb* fields are defined for HSG format only. They specify the logical block number at which this particular volume descriptor begins in least-significant-byte-first and most-significant-byte-first order, respectively.

The *cdf_vold_type* field specifies the volume descriptor type. This should be one (1) for both formats.

The *cdf_std_id* field gives 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 usually used to differentiate between the two formats.

The *cdf_vold_version* field gives the volume descriptor version number for the volume. This value should be one (1) for both formats.

The *cdf_sys_id* field 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.

The *cdf_vol_id* field 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.

The *cdf_vol_size_lsb* and *cdf_vol_size_msb* fields specify the size of this particular volume in logical blocks. This value is given in least-significant-byte-first and most-significant-byte-first order, respectively.

The *cdf_volset_siz_lsb* and *cdf_volset_siz_msb* fields give the volume set size of the volume set of which this volume is a member. This value is given in least-significant-byte-first and most-significant-byte-first order, respectively.

The *cdf_volset_seq_lsb* and *cdf_volset_seq_msb* fields give the volume sequence number of this volume. This value is given in least-significant-byte-first and most-significant-byte-first order, respectively. The volume sequence number determines the order in which volumes occur in the volume set.

The *cdf_logblk_siz_lsb* and *cdf_logblk_siz_msb* fields give the logical block size for this volume. This value is given in least-significant-byte-first and most-significant-byte-first order, respectively. The logical block size cannot be less than 512 bytes, and cannot exceed the logical sector size of the device (which currently does not exceed 2048 bytes).

The *cdf_pathtbl_siz_lsb* and *cdf_pathtbl_siz_msb* fields give the size of the path table for this volume in bytes. This value is given in least-significant-byte-first and most-significant-byte-first order, respectively. A path table defines the directory hierarchy of the file system on the volume. The first entry in the table describes the root directory.

The *cdf_pathtbl_loc_lsb* field gives the 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.

The *cdf_pathtblo_loc_lsb* field (in ISO format) and *cdf_pathtblo1_loc_lsb*, *cdf_pathtblo2_loc_lsb*, and *cdf_pathtblo3_loc_lsb* fields (in HSG format) 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.

The *cdf_pathtbl_loc_msb* field gives the 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.

The *cdf_pathtblo_loc_msb* field (in ISO format) and the *cdf_pathtblo1_loc_msb*, *cdf_pathtblo2_loc_msb*, and *cdf_pathtblo3_loc_msb* fields (in HSG format) 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.

The *cdf_rootdp* field is a 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];
```

};

The meanings of the fields in this structure are:

mincdd_reclen	Length of this directory record, in bytes. This will always be 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 will be 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 will be the block where the XAR begins; otherwise, this will be 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, only 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 will always be zero for the root directory).
mincdd_lg_size	Size of the interleave gap if the file is interleaved (this field will be 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* above, 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 will always

be 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.

The *cdf_vol_set_id* field is a character string giving the identification of the volume set of which this volume is a member. Up to 128 characters can be specified. This ID will be left-justified in the field, and padded on the right with spaces.

The *cdf_pb_id* field contains an 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.

The *cdf_pp_id* field contains an 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.

The *cdf_ap_id* field contains an 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 will 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.

The *cdf_copyright* field 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.

The *cdf_abstract* field 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 will be left-justified in the field, and padded on the right with spaces.

The *cdf_bibliographic* field is defined in ISO format only and 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.

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.

The *cdfs_fs_version* field specifies the version number of the specification of the directory and path table records. For both formats, this value is one (1).

The *cdf_appl_use* field is 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.

The *cdf_future_use* field is reserved for future use, and will be initialized to contain zeros. This field contains 653 bytes in ISO format, and 680 bytes in HSG.

NOTES

At the current time, only the *statfs*(2) system call returns information from the volume descriptor, and then only the logical block size and the size of the volume are returned. The rest of the volume descriptor can only be read via a raw read of the CD-ROM itself.

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 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;
   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[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:

```
#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*(2) system call attempts to map whatever information is included in the cdnode for a given file into the standard stat structure. 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 release of HP-UX. 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.

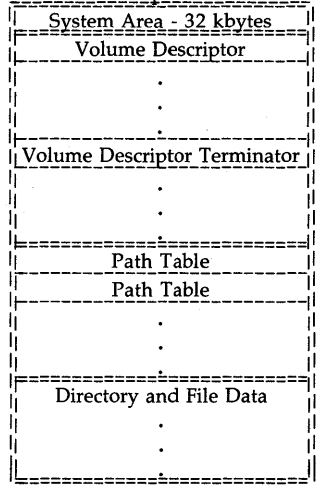
Standard Formats

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 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 re-work the entire set.

Volume Descriptors

Each directory hierarchy on a volume is described by aa *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*(2) system call. 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*(2). The user can request the XAR for a particular file or directory with the *fsctl*(2) system call.

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 may 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 may appear more than once in a single file, or appear many times in many different files. A file section in one volume may also be claimed by a file in a subsequent volume in a volume set (this is the way 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.

Directories must always consist of a single section.

Refer to *cdfsdir*(4) for more information.

Implementation and Interchange Levels

The CD-ROM standards define two levels of implementation and three levels of interchange. *implementation levels* provide a way for receiving systems which 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).

The *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

checklist – static information about the file systems

SYNOPSIS

#include <checklist.h>

DESCRIPTION

Checklist is an ASCII file that resides in the 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. The 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* will be sequentially checked by *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

The entries from this file are accessed using the routines in *getmntent*(3X).

The fields are separated by white space, and a **#** as the first non-white character indicates a comment.

special file name is a block special file name. This field is used by the *fsck*(1M), *mount*(1M) and *swapon*(1M) 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 the *type* is **swap**, the *special file name*, is made available as a piece of swap space by the *swapon*(1M) command. The fields *directory*, *options*, *pass number*, and *backup frequency* are ignored for **swap** entries. If the *type* is **swapfs** , the file system which *directory* resides in is made available as swap space by the *swapon*(1M) command. 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*(1M) and *fsck*(1M) commands. *Fsck* will also ignore 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). *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*(1M) 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 will be ignored by the *fsck*(1M) command. If a *pass number* is not present, *fsck* will check 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 pound sign (#) and ends with a newline. Space from the *pass number* up to the *comment* field, if present, or the newline 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 fields *pass number*, and *backup frequency* are ignored for NFS entries.

EXAMPLES

Examples of entries specified in **/etc/checklist**:

To add an additional file system:

/dev/dsk/c0d1s0 /users hfs defaults 1 0 # /users disk

To add a swap device:

/dev/dsk/c0d1s0 / swap defaults 1 0 # swap device

To add file system swap space:

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

EXAMPLES

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 Bell Laboratories, 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 is reserved for future use. It should be empty.

A description of each cluster node (detailed below).

A cluster node is described by a series of colon (:) separated fields, terminated by a newline 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 root server is 1.
cnode name	The name associated with this cnode of the cluster. The cnode name may 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 root server, the character will be 'r'; otherwise, it will be 'c'.
swap location	If this machine uses the root server's swap space, this will be the cnode ID of the root server. If swapping locally, it will be the cnode ID of itself.
csp	The default number of kernel level server processes to create when the *csp*(1M) command is executed.

The file **/etc/clusterconf** is usually accessed by the routines *getccent*, *getccmid*, *getccnam*, *setccent*, *endccent*, and *fgetccent*. These routines are documented on *getccent*(3C).

The *cct_entry* structure defined in **<cluster.h>** is defined as follows:

```
struct cct_entry {
    u_char machine_id[6];          /* Machine ETHERNET address */
    cnode_t cnode_id;              /* cnode ID */
    char cnode_name[15];           /* cnode name */
    char cnode_type;               /* 'r' for root server,
                                      'c' for all others */
    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

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, then 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 back in place is performed to break the tie. The case of letters may also be used in this fashion.

This table has four sections: a file header, a sequence table, a two_to_one mapping table, and a one_to_two mapping table.

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

Entries in the sequence table 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, then 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).

NAME

core – format of core image file

DESCRIPTION

The HP-UX system writes out a core image of a terminated process when any of various errors occur. See *signal*(5) for the list of reasons; the most common are memory violations, illegal instructions, floating point exceptions, bus errors, and user-generated quit signals. The core image is called **core** and is written in the process's working directory (provided it can be; normal access controls apply). A process with an effective user ID different from the real user ID will not produce a core image.

The file contains sufficient information to determine what the process was doing at the time of its termination. Debuggers such as *cdb*(1) and *adb*(1) provide a uniform user interface to this information. The contents of the file are implementation-dependent; refer to the DEPENDENCIES section below for further details.

DEPENDENCIES

Series 300

The first section of the core image is a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. The size of this section depends on the parameter UPAGES, which is defined in <**sys/param.h**>. The remainder represents the contents of the user's core area when the core image was written. If the text segment is read-only and shared, or separated from data space, it is not dumped.

The format of the information in the first section is described by the *user* structure of the system, defined in <**sys/user.h**>. The locations of the registers, however, are specified in <**machine/reg.h**>.

Series 800

The first section of the core image is a copy of the system's per-user data for the process. This includes the *user* structure, the registers and the kernel stack. The size of this section depends on the parameter UPAGES, which is defined in <**sys/param.h**>. The UPAGES block consists of USIZE pages of the *user* structure followed by STACKSIZE pages of the kernel stack. The *user* structure is defined in <**sys/user.h**>. Additional information about this section of the core image can be found in <**machine/reg.h**> and <**machine/save_state.h**>.

The remainder of the core image represents the contents of the user's core area when the core image was written (that is, the user data area followed by the user stack). The text segment is always read-only and shared; thus, it is never dumped.

The Series 800 does not support *cdb*(1); instead, see *xdb*(1) for a similar interface.

SEE ALSO

adb(1), cdb(1), setuid(2), signal(5).

NAME

cpio − format of cpio archive

DESCRIPTION

The *header* structure, when the **−c** option of *cpio*(1) is not used, 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 **−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(2)*, 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 -x), c_rdev will contain a magic constant which is dependent upon the implementation which is doing the writing. *H_rdev* flags the device file as an HP-UX 32-bit device specifier, and c_filesize will contain the 32-bit device specifier (see *stat(2)*). If the -x option is not present, special files are not archived or restored. Non-HP-UX device special files are never restored.

SEE ALSO

cpio(1), find(1), stat(2).

STANDARDS CONFORMANCE

cpio: XPG2, XPG3, POSIX.1, FIPS 151-1

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*(1M). Normally, this file resides in the directory **/etc**.

This is an ASCII file consisting of zero or more lines where each line is terminated by a newline 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, c_major. They represent logical unit number, hardware address, block major number, character major number, respectively. Parameters may appear in any order after the name; however, they must be directly preceded by their keyword.

For example, lines from a *devices* file follow:

```
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*(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*(3). 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*(1) will request an additional password, and check 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 data base which describes disk geometries and disk section characteristics. Entries in *disktab* consist of a number of ':' separated fields. The first entry for each disk gives the names which are known for the disk, separated by '|' 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>** on your system.

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

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).

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*(1) cannot be used on a DOSIF volume because the operating system does not recognize it.

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.

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

fs – format of file system volume

SYNOPSIS

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/fs.h>
#include <sys/ino.h>
#include <sys/inode.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 which may be free. If the freelist drops below this level only the super-user may continue to allocate blocks. This may be set to 0 if no reserve of free blocks is deemed necessary, however severe performance degradations will be observed 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).

N.B.: 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.

N.B.: 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/i-number pair. For the format of an inode and its flags, see *inode*(4).

DEPENDENCIES

Series 300

The series 300 supports 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.

Series 800

Continuation inodes are not implemented.

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). Such files must generally 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 the 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. The canned tabs which are recognized are defined by the *tabs*(1) command.

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 prepended.

m*margin* The **m** parameter specifies a number of spaces to be prepended to 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*(1), 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

gettydefs – speed and terminal settings used by getty

DESCRIPTION

The **/etc/gettydefs** file contains information used by *getty*(1M) to set up the speed and terminal settings for a line. 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*(2) settings to which the terminal is to be set if a terminal type is not specified to *getty*. 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*(1).

final-flags These flags take the same values as the *initial-flags* and are set just prior to *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, then *getty* will search 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 instance, **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, then 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* can not find the specified *label*. If **/etc/gettydefs** itself is missing, there is one entry built into the command which will bring 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 be sure 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

group – 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. The fields are separated by colons; each group is separated from the next by a new-line. 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 via *newgrp*(1). **/etc/logingroup** provides a default group access list for each user via *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 ID's 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, they can and do have general read permission and can be used, for example, to map numerical group ID's to names.

Grp.h describes the group structure returned by *getgrent*(3C), etc:

```
/* see getgrent(3C) */
    struct      group {
                    char        *gr_name;
                    char        *gr_passwd;
                    int         gr_gid;
                    char        **gr_mem;
    };
```

NETWORKING FEATURES

NFS

The **/etc/group** file can have a line beginning with a plus (+), which means to incorporate entries from the yellow pages. There are two styles of + entries: + means to insert the entire contents of the yellow pages group file at that point, and +*name* means to insert the entry (if any) for *name* from the yellow pages at that point. If a + entry has a non-null password or group member field, the contents of that field will overide what is contained in the yellow pages. 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 - entries: An entry that consists of -*name* means to disallow any subsequent entry (if any) for *name*. These entries will be disallowed regardless of whether the subsequent entry comes from the yellow pages or the local group file.

WARNINGS

The gid 9 is reserved for the Pascal Language operating system and the BASIC Language

operating system. These are operating systems for the Series 300 computers that can co-exist with HP-UX on the same disk. Using this gid for other purposes can inhibit file transfer and sharing.

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
+:
```

The group *other* will have a gid of 1 and members *root*, *daemon*, *uucp*, *who*, *date*, and *sync*. The group *oldproj* will be ignored since it appears after the entry *-oldproj*. Also, the group *myproject* will have members *bill* and *steve*, and the password and group ID of the yellow pages entry for the group *myproject*. All the groups listed in the yellow pages will be 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 will not work.

FILES

/etc/group
/etc/logingroup

SEE ALSO

groups(1), newgrp(1), passwd(1), setgroups(2), crypt(3C), getgrent(3C), initgroups(3C), passwd(4).

BUGS

There is no single tool available to completely ensure that **/etc/passwd**, **/etc/group**, and **/etc/logingroup** are compatible. However, *pwck*(1M) and *grpck*(1M) can be used to simplify the task.

There is no tool for setting group passwords in **/etc/group**.

STANDARDS CONFORMANCE

group: SVID2, XPG2

NAME

inittab – script for the init process

DESCRIPTION

The *inittab* file supplies the script to *init*'s role as a general process dispatcher. 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*(1) command. 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.

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 levels* 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* **0–6**.

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 upon entering a new *run level* where 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 will request 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 legal *sh* syntax can appear in the *process* field. Comments can be inserted with the **;** **#***comment* syntax.

In the HP Clustered environment, **/etc/inittab** is a context dependent file (CDF) since different cnodes have different initialization requirements. See *cdf*(4).

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. It 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.

DEPENDENCIES

Series 800

The *di_fversion* field is not used.

Continuation inodes are not currently implemented.

FILES

/usr/include/sys/ino.h

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 program *getty* and 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 disk 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, and 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.

The LIF utility on HP-UX currently supports three file types, ASCII (1), BINARY (-2) and BIN (-23951).

Three copying modes are associated with them:

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 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 will have 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 media. 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, the 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*(2). Note that you should not mount the special file before using the *lif**(1) utilties. 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 within 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 legal path naming for any other use within the HP-UX operating system.

SEE ALSO

lifcp(1), lifinit(1), lifls(1), lifrename(1), lifrm(1).

NAME

magic – magic numbers for HP-UX implementations

SYNOPSIS

#include <magic.h>

DESCRIPTION

Magic.h localizes all information about HP-UX "magic numbers" in one file, and thus facilitates 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 HPE_MAGIC       0x150   /* HPE boot image */
```

The values for *system_id* are defined in *model*(4).

WARNINGS

Cpio files use a different form of magic number that is incompatible with *magic*(4).

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*(1M) to obtain device information that enables it to generate the configuration file.

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: Ten bits make up the mask

- Bit 1 - size handler
- Bit 2 - link routine
- Bit 3 - open handler
- Bit 4 - close handler
- Bit 5 - read handler
- Bit 6 - write handler
- Bit 7 - ioctl handler
- Bit 8 - select handler
- Bit 9 - seltru handler
- Bit 10 - 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 daemon *mirlog*(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

```
/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

mknod – create a special file entry

SYNOPSIS

#include <sys/mknod.h>

DESCRIPTION

Mknod.h provides utilities to pack and unpack device names as used by *mknod*(2). It contains the macro **dev = makedev(major, minor)** which packs the major and minor fields into a form suitable for *mknod*(2). It also contains **major(dev)** and **minor(dev)** which extract the corresponding fields.

The macro MINOR_FORMAT is a *printf* specification that prints the minor field in the format best suited to the particular implementation. The specification given by MINOR_FORMAT must cause the resulting string to indicate the base of the number in the same format as that used for C: no leading zero for decimal, leading zero for octal, and leading zero and 'x' for hexadecimal.

When a minor field is printed in the format specified by MINOR_FORMAT, each sub-field contained in the minor will be wholly contained in the mininum possible number of digits of the resulting string. (Splitting a field across unnecessary digits for the sake of packing is not done.)

SEE ALSO

mknod(1M), mknod(2).

NAME

mnttab – mounted file system table

SYNOPSIS

#include <mnttab.h>

DESCRIPTION

Mnttab resides in directory **/etc** and contains a table of devices, mounted by the *mount*(1M) command. The file contains a line of information for each mounted filesystem, structurally identical to the contents of **/etc/checklist**, described in *checklist*(4).

There are a number of lines of the form:

special_file_name dir type opts freq passno mount_time

For example:

/dev/dsk/c0d0s0 / hfs rw 0 1 537851723

The file is accessed by programs using *getmntent*(3X), and by the system administrator using a text editor.

Mount_time contains the time the file system was mounted using *mount*(1M). The value is the number of seconds since 00:00:00 GMT, January 1, 1970; see *time*(2).

WARNINGS

The table is provided only as a means for programs to return information about mounted file systems. Duplicate or missing entries in *mnttab* cannot impair the correct operation of *mount* and *umount*.

Mnttab is initialized when the system starts multi-user operation, and may not accurately reflect what was mounted while in single-user mode if the system was first booted as a single-user system instead of multi-user.

AUTHOR

Mnttab was developed by the University of California, Berkeley, Sun Microsystems, Inc., and HP.

FILES

/etc/mnttab

SEE ALSO

mount(1M), setmnt(1M), getmntent(3X), checklist(4).

NAME

model – HP-UX machine identification

SYNOPSIS

#include <model.h>

DESCRIPTION

There are some distinctions between the implementations of HP-UX 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 *model.h*. This file contains constants identifying various HP-UX implementations.

For example the header file *model.h* contains the following constants for the HP 9000 Series 300.

```
/* model.h for the HP 9000 Series 300 */

#define     HP_S_300     0x20A
#define     HP_S_500     0x208
#define     HP_S_800     0x20B
```

Other such constants will be added as HP-UX extends to other machines.

In addition, *model.h* has a statement defining the preprocessor constant *MYSYS* to represent the specific implementation for which compilation is desired. *MYSYS* will be equal to one of the constants above.

Conditional compilation may be used to adapt one file for execution on more than one HP-UX implementation, if it contains implementation- or architecture-dependent features. For instance,

```
#if MYSYS==HP_S_300
        <statements>
#endif
```

will cause the statements following the if statement to be compiled only for the HP 9000 Series 300.

Model.h also contains typedefs for several predefined types to enhance portability of certain types of code and of 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

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 the various implementations of HP-UX.

DESCRIPTION

Nlist(3C) can be used to extract information from the symbol table in an object file. Because 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 for each user the following information:

login name
encrypted password
numerical user ID
numerical group ID
reserved field, which may 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. 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.

This file resides in directory **/etc**. It can and does have general read permission and can be used, for example, to map numerical user IDs to names.

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 effected 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 – such as *M* – denotes the maximum number of weeks for which a password is valid. A user who attempts to login after his password has expired will be forced to supply a new one. The next character – such as *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 will be forced to change his password next time he logs in (and the "age" will disappear 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 may 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 with the *chfn*(1) command 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 may be represented in the **pw_gecos** field by an **&** character, which some utilities (including *finger*(1)) expand by substituting the login name for it and then 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** this 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 Yellow Page network database. A line beginning with a plus (+) is used to incorporate entries from the Yellow Pages. There are three styles of + entries: all by itself, + means to insert the entire contents of the Yellow Pages password file at that point; +*name* means to insert the entry (if any) for *name* from the Yellow Pages at that point; +*@name* means to 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 will override what is contained in the Yellow Pages. 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 Yellow Pages. There are two styles of - entries: *-name* means to disallow any subsequent

entries (if any) for *name*, and -*@name* means to disallow any subsequent entries for all members of the network group *name*.

WARNINGS

The uid 17 is reserved for the Pascal Language operating system. The uid 18 is reserved for the BASIC Language operating system. These are operating systems for the Series 300 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. The 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.

DEPENDENCIES

Series 300

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.

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/csh
+john:
-bob:
+@documentation:no-login:
-@marketing:
+:::Guest
```

In this example, there are specific entries for users **root** and **joeuser**, in case the Yellow Pages are out of order. The user **john** will have his password entry in the Yellow Pages incorporated without change; any subsequent entries for the user **bob** will be ignored; anyone in the netgroup **documentation** will have their password field disabled; anyone in the netgroup **marketing** will not be returned by *getpwent*(3C) and thus not allowed to login, and anyone else will be able to 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 passwds can be accessed only in **/.secure/etc/passwd**. Any user entry in the Yellow Pages database also must have an entry in **/.secure/etc/passwd**.

The uid of -2 is reserved for remote root access with NFS. The **pw_name** usually given to this uid is **nobody**. Since uid are stored as unsigned values, the following define is included in **pwd.h** to match the user **nobody**.

```
#define  UID_NOBODY  ((ushort) 0xfffe)
```

SEE ALSO
netgroup(4).

FILES
/etc/passwd

SEE ALSO
chfn(1), finger(1), login(1), passwd(1), a64l(3C), crypt(3C), getpwent(3C), group(4).

STANDARDS CONFORMANCE
passwd: SVID2, XPG2

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 (:).

Pathname is the 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 and *group* are either the symbolic or numeric *ID*s of the owner and group of the file.

Mode is the symbolic representation of the permission and type information of the file as displayed by the **ls** –**l** command.

Size is the 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 is the number of hard links to *pathname*.

Version is the numeric value of the revision of the file. Commands supporting *PDF*s determine this value by invoking the *what(1)* command on the file and searching for a revision number. If no revision is found, the *ident(1)* command is invoked. The version number recorded is the first one encountered. If no version number is found, the field is empty.

Checksum is the result of the application of the POSIX checksum algorithm to the file's contents. This checksum is identical to that produced with the **sum** –**p** command.

Linked_to is the 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*(1M) and *pdfck*(1M)) 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*. 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 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

```
% Product Description File
% fileset TEST, Release 1.0
```

```
/bin/basename:bin:bin:-r-xr-xr-x:4904:1:56.1:52902:
/bin/cat:bin:bin:-r-xr-xr-x:27724:1:51.1:1665:
/bin/cc:bin:bin:-r-xr-xr-x:32664:1:63.3:38478:
/bin/dirname:bin:bin:-r-xr-xr-x:4892:1:51.2:2339:
/bin/grep:bin:bin:-r-xr-xr-x:35344:1:62.1:53539:
/bin/ls:bin:bin:-r-xr-xr-x:94208:6:64.1:30408:
/bin/ll:bin:bin:-r-xr-xr-x:94208:6:64.1:30408:/bin/ls
/bin/su:root:other:-r-sr-xr-x:72652:1:64.3:39793:
% total size is 272388 bytes.
% total size is 266 blocks.
```

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 Hewlett-Packard Company.

FILES

/system/*/pdf

SEE ALSO

mkpdf(1m), pdfdiff(1m), pdfck(1m).

NAME

privgrp – format of privileged values

SYNOPSIS

#include <sys/privgrp.h>

DESCRIPTION

Setprivgrp(2) sets a mask of privileges, and *getprivgrp*(2) returns an array of structures giving privileged group assignments on a per group–id basis. *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

/*
 * Size of the privilege mask,
 * based on largest numbered privilege
 */
#define PRIV_MASKSIZ  1

/*
 * Structure defining the privilege mask
 */
struct privgrp_map {
   int    priv_groupno;
   unsigned int priv_mask[PRIV_MASKSIZ];
};
```

PRIV_RTPRIO	allows access to the *rtprio*(2) system call.
PRIV_MLOCK	allows access to the *plock*(2) system call.
PRIV_CHOWN	allows access to the *chown*(2) system calls.
PRIV_LOCKRDONLY	permits the use of the *lockf*(2) system call for setting locks on files open for reading only.
PRIV_SETRUGID	permits the use of the *setuid*(2) and *setgid*(2) system calls for changing respectively the real user ID and real group ID of a process.

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 may have several different privileges associated with it by having different bits **or**'ed into the mask.

The system is configured with a 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 defining privileges associated with a group–id.

Privileges are returned to the user from the *getprivgrp*(2) 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 your login (home) directory contains a file named **.profile**, that file will be executed (via the shell's **exec .profile**) before your 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 $TERM in
        300)            stty cr2 nl0 tabs; tabs;;
        300s)           stty cr2 nl0 tabs; tabs;;
        450)            stty cr2 nl0 tabs; tabs;;
        hp)             stty cr0 nl0 tabs; tabs;;
        745|735)        stty cr1 nl] -tabs; TERM=745;;
        43)             stty cr1 nl0 -tabs;;
        4014|tek)       stty cr0 nl0 -tabs ffl; TERM=4014; echo "33;";;
        *)              echo "$TERM unknown";;
esac
```

A more complete model **.profile** may 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*(1) or *batch*(1), the job is constructed as a Bourne shell script. The job file is created in **/usr/spool/cron/atjobs** as follows:

- *At*(1) creates a header describing the job as an **at** job or a **batch** job. *At*(1) jobs submitted to all queues other than queue **a** are listed as **batch** jobs. The header will be
  : at job

  for an **at** job, and
  : batch job

  for a **batch** job.
- A set of Bourne shell commands is added to make the environment (see *environ*(5)) for the *at*(1) job the same as the current environment.
- *At*(1) then copies text from the prototype file to the job file, except for special variables that are replaced by other text:

  $d is replaced by the current working directory
  $l is replaced by the current file size limit (see *ulimit*(2))
  $m is replaced by the current umask (see *umask*(2))
  $t is replaced by the time at which the job should be run, expressed as seconds since January 1, 1970, 00:00 Greenwich Mean Time, preceded by a colon
  $< is replaced by text read by *at*(1) from the standard input (that is, the commands provided to *at*(1) to be run in the job)

When a job is submitted to queue *queue*, *at*(1) will use the file **/usr/lib/cron/.proto.***queue* as the prototype file if it exists, otherwise it will use the file **/usr/lib/cron/.proto**.

EXAMPLES

The following **.proto** file:

```
cd $d
ulimit $l
umask $m
$<
```

creates commands to change the current directory in the job to the current directory at the time *at*(1) was run, to change the file size limit in the job to the file size limit at the time *at*(1) was run, and to change the umask in the job to the umask at the time *at*(1) was run. These commands are inserted before the commands in the job.

SEE ALSO

at(1), queuedefs(4)

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*(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*(1), **b** is the queue for jobs started by *batch*(1), and **c** is the queue for jobs run from a *crontab*(1) file. Queue names **d** through **y** designate user defined queues.

njob The maximum number of jobs that can be run simultaneously in that queue. The default value is 100.

nice The *nice*(1) value to give to all jobs in that queue that are not run with a user ID of super-user. 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 25 jobs were running in all the queues. The default value is 60.

EXAMPLES

The following **queuedefs** file:

a.4j1n
b.2j2n90w

specifies that the **a** queue, for *at*(1) jobs, can have up to 4 jobs running simultaneously, and that those jobs will be run with a *nice*(1) value of 1. As no *nwait* value was given, if a job cannot be run because too many other jobs are running, *cron*(1M) will wait 60 seconds before trying again to run it. The **b** queue, for *batch*(1) jobs, can have up to 2 jobs running simultaneously. Those jobs will be run with a *nice*(1) value of 2. If a job cannot be run because too many other jobs are running, *cron*(1M) 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*(1) value of 2, and if a job cannot be run because too many other jobs are running, *cron*(1M) will wait 60 seconds before trying again to run it.

SEE ALSO

at(1), batch(1), nice(1), crontab(1), cron(1M), proto(4)

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 *ld* 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*.

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*(2) or *fseek*(3).

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 file

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 lines have no significance except in strings. Strings are enclosed by "@". If a string contains the "@" symbol, the symbol must be doubled.

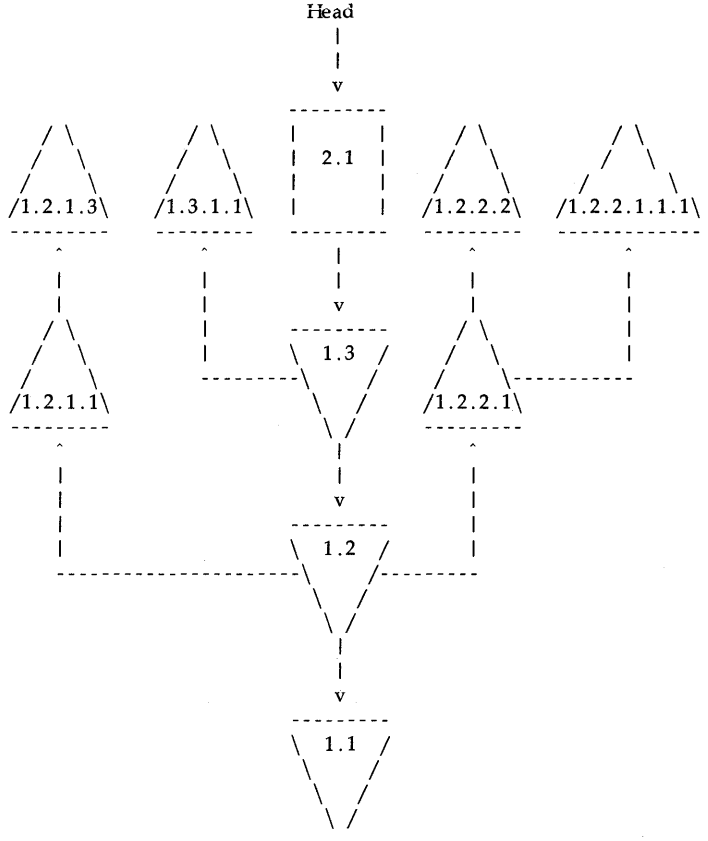
The meta syntax uses the following conventions: "|" (bar) separates alternatives; "{" and "}" enclose optional phrases; "{" and "}*" enclose phrases that may be repeated zero or more times; "{" and "}+" enclose phrases that must appear at least once and may be repeated; "<" and ">" enclose nonterminals.

```
<rcstext>     ::=  <admin> {<delta>}* <desc> {<deltatext>}*

<admin>       ::=  head          {<num>};
                   access        {<id>}*;
                   symbols       {<id> : <num>}*;
                   locks         {<id> : <num>}*;
                   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:



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), rcsintro(5), rcsmerge(1), rlog(1).

NAME

sccsfile – format of SCCS file

DESCRIPTION

An SCCS file is an ASCII file. It consists of six logical parts: the *checksum*, the *delta table* (contains information about each delta), *user names* (contains login names and/or numerical group IDs of users who may add deltas), *flags* (contains definitions of internal keywords), *comments* (contains arbitrary descriptive information about the file), and the *body* (contains the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines which begin with the **ASCII SOH** (start of heading) character (octal 001). This character is hereafter referred to as *the control character* and will be 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).

Each logical part of an SCCS file is described in detail below.

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

@hDDDDD

The value of the checksum is the sum of all characters, except those of the first line. The **@h** provides a *magic number* consisting of the two bytes 0x01 and 0x68. (Other versions of the UNIX operating system 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> <**SCCS ID**> 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 SCCS ID of the delta, the date and time of creation of the delta, 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 may 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 succeeding 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 **t** flag defines the replacement for the **%Y%** identification keyword. The **v** flag controls prompting for **MR** numbers in addition to comments; if the optional text is present it defines an **MR** number validity checking program. The **i** flag controls the warning/error aspect of the "No id keywords" message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message will cause a "fatal" error (the file will not be gotten, or the delta will not be made). When the **b** flag is present the **−b** keyletter may be used on the *get* command to cause a branch in the delta tree. The **m** flag defines the first choice for the replacement text of the **%M%** identification keyword. The **f** flag defines the "floor" release; the release below which no deltas may be added. The **c** flag defines the "ceiling" release; the release above which no deltas may be added. The **d** flag defines the default SID to be used when none is specified on a *get* command. The **n** flag 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 (e.g., 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. The **j** flag causes *get* to allow concurrent edits of the same base SID. The **l** flag defines a *list* of releases that are *locked* against editing (*get*(1) with the **−e** keyletter). The **q** flag defines the replacement for the **%Q%** identification keyword. The **z** flag is used in certain specialized interface programs.

Comments

Arbitrary text is surrounded by the bracketing lines **@t** and **@T**. The comments section

typically will contain a description of the file's purpose.

Body

The 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: *insert*, *delete*, and *end*, represented by:

@I DDDDD
@D DDDDD
@E DDDDD

respectively. The digit string is the serial number corresponding to the delta for the control line.

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. This format is based upon the format used in the Series 500 Basic workstations.

The utilities listed under SEE ALSO below are provided for non-Series 500 access to the 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 HP-UX, an SDF volume is simply a file containing unspecified data. Therefore, to access SDF media on any HP-UX system other than the Series 500, *mount*(1M) cannot be used because the operating system cannot recognize it.

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 HP-UX. The shell "wild–card" characters *, ?, and [...] do not work for specifying an arbitrary pattern for matching SDF file names when using the SDF utilities.

If the device name and a trailing colon are specified *without* a file or directory name following (for example **/dev/rdsk/c5d1s2:**), then 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, Hewlett–Packard does not support the use of the SDF utilities on SRM workstation storage media.

WARNINGS

The 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, *you are cautioned to not run them on a disk that has a mounted file system on it.*

AUTHOR

Sdf(4) was developed by the Hewlett-Packard Company.

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), sdffls(1), sdfmkdir(1), sdfrm(1).

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 replace the symbolic link component and is 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 would be interpreted as **/a/b/../x/y/d**. If, instead, **c** were 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 following system calls: *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 may refer to any arbitrary path name and may span different logical devices (volumes). The path name may be that of any type of file (including a directory, or another symbolic link) or it may 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.

Symbolic links can be created using *ln* (on *cp*(1)) or *symlink*(2).

NETWORKING FEATURES

RFA

When a symbolic link that refers to an absolute path name is located on a remote node (that is, using the Remote File Access (RFA) facility; see *mknod*(1M)), the path name is interpreted relative to the root of the remote node, not the node on which the process is executing.

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 of *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, 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 will notify the user of the error, and no attempt will be made to store any part of the file, header or data, on the medium.

In the HP Clustered environment, CDF's are located by appending a '+' onto *name* and testing for a directory using stat(2). If the test fails the '+' is removed.

SEE ALSO

tar(1), tar(5)

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 will be the same on all hardware. An 8 or more bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the *tic*(1M) program, and read by the routine *setupterm*. Both of these pieces of software are part of *curses*(3X). The file is divided into six parts: the header, terminal names, Boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are:

(1) the magic number (octal 0432);
(2) the size, in bytes, of the names section;
(3) the number of bytes in the Boolean section;
(4) the number of short integers in the numbers section;
(5) the number of offsets (short integers) in the strings section;
(6) the 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, and 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.

As an example, an octal dump of the description for the HP Portable Computer (HP-110) is included:

```
110|hp110|hp110a portable computer,
    am, xhp, da, db, mir, cols#80, lines#16, lm#0,
    cbt=\Ei, 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, il1=\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 data base

SYNOPSIS

/usr/lib/terminfo/?/*

DESCRIPTION

Terminfo is a data base 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 ',' separated fields. White space after each ',' is ignored. The first entry for each terminal gives the names which are known for the terminal, separated by '|' 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 may well 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 capname 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.

(P) indicates that padding may be specified

(G) indicates that the string is passed through tparm withparms 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	cub1 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; can not print ~'s
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
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

TERMINFO(4)

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_f10,	kf10	ka	Sent by function key f10
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_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	l0	Labels on function key f0 if not f0
lab_f1,	lf1	l1	Labels on function key f1 if not f1
lab_f10,	lf10	la	Labels on function key f10 if not f10
lab_f2,	lf2	l2	Labels on function key f2 if not f2
lab_f3,	lf3	l3	Labels on function key f3 if not f3
lab_f4,	lf4	l4	Labels on function key f4 if not f4
lab_f5,	lf5	l5	Labels on function key f5 if not f5
lab_f6,	lf6	l6	Labels on function key f6 if not f6
lab_f7,	lf7	l7	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
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
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=$<16*>,
        is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E,
        kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;,
        kf1=\E5, kf2=\E6, kf3=\E7, khome=\E?,
        lines#24, mir, pb#9600, prot=\El, rep=\Er%p1%c%p2%' '%+%c$<.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 may continue onto multiple lines by placing white space at the beginning of each line except the first. Comments may be included on lines beginning with "#". Capabilities in *terminfo* are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give 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 may 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, e.g., '20', or a number followed by an '*', i.e., '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 one 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 decimal place is allowed.)

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 a control-x for any appropriate x, and the sequences **n** **l** **r** **t** **b** **f** **s** give a newline, linefeed, return, tab, backspace, formfeed, and space. Other escapes include \^ for ^, \\ for \, \, for comma, \: for :, and \0 for null. (\0 will produce \200, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given 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

We now outline 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 to build up a description gradually, 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 bugs in *vi*. To easily test a new terminal description you can set the environment variable TERMINFO to a pathname of a directory containing the compiled description you are working on and programs will 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 hit the 'u' key several times quickly. If the terminal messes up, 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, then 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, then 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 this will be carriage return, control 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 will go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The 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** (newline). 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 glass-tty 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, using two parameters: 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.

The % 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 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
%+ %- %* %/ %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 ala 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 one would 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 which 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**, as the system may change or discard them. (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) then 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, then 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. The cursor position is, alas, 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, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character which 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 which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type abc def using local cursor motions (not spaces) between the abc and

the def. Then 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 to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the abc shifts over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for insert null. While these are two logically separate attributes (one line vs. multiline insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

Terminfo can describe both terminals which have an insert mode, and terminals which 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 will not give **ich1**; terminals which 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 both to 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 which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an 'insert mode' and a special code to 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*, will repeat the effects of **ich1** *n* times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** will affect 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 your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should 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, then **xmc** should be given to tell 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, as 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, which 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, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then 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 like 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 your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then 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, ..., f10, the codes they send can be given as **kf0, kf1, ..., kf10**. 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 will be 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** will 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, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. 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 will be 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, and 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.

Glitches

Hazeltine terminals, which do not allow '~' characters to be displayed should indicate **hz**.

Terminals which 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", that 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 control C. (Only certain Superbees have this problem, depending on the ROM.)

Other specific terminal problems may be corrected by adding more capabilities of the form **x**x.

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 only supports 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 *HP-UX Concepts and Tutorials: Device I/O and User Interfacing.*

STANDARDS CONFORMANCE

terminfo: SVID2

NAME

ttytype – data base of terminal types by port

SYNOPSIS

/etc/ttytype

DESCRIPTION

Ttytype is a database containing, for each tty port on the system, the kind of terminal that is attached to that port. There is one line per port, containing the terminal kind (as a name listed in *terminfo*(4)), a space, and the name of the tty, less the initial "/dev/". For example, for an HP 2622 terminal on tty02:

```
2622 tty02
```

This information is read by *tset*(1) and by *login*(1) to initialize the TERM variable at login time.

AUTHOR

Ttytype was developed by the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

login(1), tset(1).

BUGS

Some lines are merely known as "dialup" or "plugboard".

NAME

tztab – time zone adjustment table for date(1) and ctime(3C)

DESCRIPTION

The *tztab* file describes the differences between Greenwich Mean Time (GMT) 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 GMT, 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 will detail 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 (1970-1999), 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 may 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 GMT. **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 American pattern.

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 1987–1999 0 EDT4
0 1 24–30 11 1974 0 EST5
0 1 25–31 10 1975–1999 0 EST5
```

Normally (as indicated in the first line) Eastern Standard Time is five hours earlier than GMT. 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 Hewlett-Packard Company.

FILES

/usr/lib/tztab

SEE ALSO

date(1), ctime(3C), environ(5).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

NAME

update – update-media format

DESCRIPTION

Update media consists of a simple *tar*(1) archive with a few leading information files used by *update*(1M) plus specially-crafted file paths that allow files to be grouped into filesets. The following is intended as an aid in interpreting update media; not as a design guide for building your own.

Information Files

The information files contain ASCII text and include:

system/INDEX This file starts with a header line describing the general title of the update media and the number of this media unit (such as in multiple-tape sets). The rest of the file is a list of fileset names, the names of the partitions to which the filesets belong, specification of dependent filesets, and the media (tape) number on which each fileset is found. 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 has essentially two parts. The first part lists the filesets along with a description of each fileset, flags associated with each fileset (see Fileset Flags below), and the media number on which that fileset resides. This media number is ignored in deference to the media number in the INDEX file.

The second part of the file lists, for each fileset on the update media set, the files in the fileset and their sizes. 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. *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.

Fileset Flags

Each fileset has five flags associated with it. Their meanings are:

B Rebuild the kernel and reboot the system.

C Change of destination is not allowed for this fileset.

D Delay running of customization scripts for all filesets loaded.

H Fileset contains HP-PA (Series 800) files, as opposed to HP-MC68020 (Series 300) files.

Y Use sysrm(1M) to remove the files in the fileset before loading new ones.

The **B**, **C**, **D**, and **Y** flags are used by *update*(1M) and ignored by *updist*(1M).

In addition, the **P** flag appears on partition description lines only.

Path Names

On update media, path names of files other than the information files consist of their official path names with a concatenated prefix:

fileset/..

For example:

CORE/../bin/sh

These path names indicate each file's fileset and still allow unpacking with *tar* relative to any directory. Unpacking an update media unit with *tar* rather than *update* 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).

File Types

Update (*tar*) media support the following types of files: ordinary, directory, symbolic link, character special, block special, and FIFO. Network special files and sockets are not supported. Furthermore, *update* and *updist* do not support creation of character special, block special, and FIFO files.

EXAMPLES

To list the files on an update media unit accessed through special file **/dev/rmt**:

tar −tvf /dev/rmt

To study the leading information files on that update media unit:

```
cd /tmp
tar −xvf /dev/rmt system
more system/*
```

SEE ALSO

tar(1), sam(1M), sysrm(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*(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

acctcon(1M), fwtmp(1M), last(1), login(1), who(1), write(1), getut(3C), cdf(4).

STANDARDS CONFORMANCE

utmp.h: XPG2

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

Remarks:

To ensure continued conformance with emerging industry standards, features described in this manual entry are likely to change in a future release.

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 and RFA. 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

RFA and NFS

ACLs are not supported on remote files by RFA or NFS. Individual manual entries specify the behavior of various system calls, library calls, and commands under these circumstances. Take care 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

Ascii is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. It contains:

```
|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

SEE ALSO

kana8(5), roman8(5).

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 id**s 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 HP-UXng 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), *vfsmount*(2), and *rfa_netunam*(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)).

The following strings in the context indicate the ability to run executable code for the designated hardware. When more than one appear in the same context, they will appear in the order listed.

HP98248A
HP-MC68881
HP98635A
HP-MC68020
HP-MC68010

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.

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) will 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

cdf(4), getcontext(2), getcontext(1).

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>**.

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: SVID2, XPG2, XPG3, POSIX.1, FIPS 151-1

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 HOME is the name of the user's login directory, set by *login*(1) from the password file (see *passwd*(4)).

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 run-time 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**>).

LANGOPTS Defines language options for mode and data order in the form:

LANGOPTS=[*mode*][*_order*]

Values of LANGOPTS are given in English as an ASCII character string. The *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**. The *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

These internationalization environment variables correspond to the *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.

The syntax for the environment variables LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, and LC_TIME is:

language[*_territory*][*.codeset*][**@***modifier*]

The 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 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**>).

NLSPATH 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) will attempt 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) will attempt 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

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: XPG2, XPG3, POSIX.1, FIPS 151-1

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: XPG2, XPG3, POSIX.1, FIPS 151-1

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, post-nroff form.

/usr/contrib/man/man[1-8]
User-contributed manual entries, pre-nroff form.

/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, post-nroff form.

/usr/local/man/man[1-8]
Site-local manual entries, pre-nroff form.

/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 (post-nroff) versions of online documentation for use by *man*(1).

/usr/man/man[1-8] Unformatted (pre-nroff) 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

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 language. NLS is available at the user command level as well as through commands and libraries that can be used to develop international software applications. HP-UX NLS is a combination of standards specified by the X/Open Portibilty Guide, Issue 2 and 3, IEEE 1003.1 and ANSI C as well as HP added enhancements.

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 the *setlocale*(3C) routine. For a complete list of what library routines are affected by *setlocale*, see *setlocale*(3C).

In addition to the routines which operate based on the program's locale, there are also commands and routines which 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 Variable" sections of each command with NLS capabilites describes which environment variables the command is sensitive to.

In addition, Portnls is a set of library routines which perform miscellaneous language-dependent operations. Portnls is intended to provide portibility between HP-UX and MPE (another HP operating system). See *portnls*(5) for more information.

Below are areas of functionality which 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. A multibyte encoding scheme, the scheme used to encode multibyte character sequences, is used to define codesets for ideographic languages. Multibyte codesets define character codes which are represented by one or more bytes. The encoding scheme used to define the multibyte codesets is described in *HP-UX Concepts and Tutorials*.

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 sensitives to these differences.

Shifting

The notion of "case" differs between languages. For example, in some languages accents are discarded when characters are shifted to uppercase. Some languages have no notion of upper and lower-case characters. For example, in ideographic languages shifting a character will have no affect.

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 comparsion :

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 (for example, "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 the first pass, accents are stripped off 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. The case of letters can also be first ignored and then used to break ties in this fashion.

Two common methods of collation for phonetic languages are folded and nonfolded. A folded collating sequence will be made up of the upper and lowercase characters intermixed. An unfolded collating sequence will be 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*(1) command will report the collating methods supported for each language.

Directionality

Two properties of text files and Native Languages must be understood to process text in non-Western languages — 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 that 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 that 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 which 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 region which 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, as well as 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 which aid the tranlation of messages, such that at runtime the program accesses messages which coincide with the end-users native language.

FILES

/usr/lib/nls/*

AUTHOR

Hpnls was developed by HP.

SEE ALSO

insertmsg(1), gencat(1), catgets(3C), catopen(3C),setlocale(3C), environ(5), lang(5)

Native Language Support, manual in *HP-UX Concepts and Tutorials: Device I/O and User Interfacing.*

For additional information, see the External Influences/Environment Variables section on other manual pages of 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 page section **(7)**. If a device does not support an ioctl request it will return **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 will not be 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*. The super-user may designate that any process receive the SIGIO signals. If the request is not made by the super-user, the calling process is only 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 another process whose real or saved effective user ID matches its real or effective user ID, or a process which is not a descendant of the calling process, the call will fail, with **errno** set to EPERM.

If the designated process subsequently exits, the SIGIO signal will not be sent to any process.

The default on open of 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 will be 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 will prevent all read requests to that device from blocking, whether the requests succeed or fail. Such a read request will complete in one of three ways: (1) If there is enough data available to satisfy the entire request, the read will complete successfully, having read all of the data, and return the number of bytes read; (2) If there is not enough data available to satisfy the entire request, the read will complete successfully, having read as much data as possible, and return the number of bytes it was able to read; (3) If there is no data available, the read will fail and **errno** will be set to EWOULDBLOCK.

For writes, non-blocking I/O will prevent all write requests to that device file from blocking, whether the requests succeed or fail. Such a write request will complete in one of three ways: (1) If there is enough space available in the system to buffer all the data, the write will complete successfully, having written out all of the data, and return the number of bytes written; (2) If there is not enough space in the buffer to write out the entire request, the write will complete successfully, having written as much data as possible, and return the number of bytes it was able to write; (3) If there is no space in the buffer, the write will fail and **errno** will be 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 will perform 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.

DEPENDENCIES

Series 300

Asynchronous I/O is not supported.

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(1)* command will display information regarding what languages are currently supported on a particular HP-UX system. In addition, *nlsinfo* will also provide information on what modifiers (see *environ*(5)) are valid for each available language.

The default processing language for HP-UX is "C". "C" provides an environment in which processing occurs without NLS functionality. This environment is based on the 7-bit US ASCII coded 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 the *nl_init*(3C) routine is used. "C" and "n-computer" are equivalent with the exception of the *nl_langinfo*(3C) items currency symbol (CRNCYSTR) and thousands seperator (THOUSEP). "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*(3C) instead.

The NLS environment can also be initialized by passing a language ID number to routines which 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 the **/usr/lib/nls/config** file. The language ID number and the routines accepting the *langid* parameter are provided for historical reasons only. Routines which provide equivalent functionality without the *langid* parameter are recommended. The WARNING section of the appropriate routine will indicate what routine to use.

AUTHOR

Lang was developed by HP.

SEE ALSO

nlsinfo(1), setlocale(3C), 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) and *strftime*(3C).
D_FMT	LC_TIME	String for formatting the **%x** (date) directive of *date*(1) and *strftime*(3C).
T_FMT	LC_TIME	String for formatting the **%X** (time) directive of *date*(1) 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).
YESSTR	LC_ALL	Affirmative response for yes/no questions.
NOSTR	LC_ALL	Negative response for yes/no questions.

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$56). See *localeconv*(3C) for alternative currency formatting information.
BYTES_CHAR	LC_CTYPE	Maximum number of bytes per character for the character set used for the specified language. For example, "1" for English and most European languages, and "2" for Japanese and several other Asian languages.
DIRECTION	LC_ALL	Value to indicate text direction. Values currently defined include "null", "0" and "1". Values of "null" or "0" indicate that characters are arranged from left-to-right within a line and lines are arranged from top-to-bottom. A value of "1" indicates that characters are arranged from right-to-left within a line and lines are arranged from top-to-bottom.
ALT_DIGITS	LC_NUMERIC	A string of the characters that are mapped into the ASCII equivalent string "0123456789b+-.,eE" (where b is a blank). This is also the reverse mapping for output. It is not assumed that the character code values of digits are contiguous or that they are one byte values. A null value for the string indicates that the language has no alternative digits.
ALT_PUNCT	LC_CTYPE	A string of the characters that are mapped into the ASCII equivalent string "b!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~" (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.
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_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 *buildlang*(ADMIN)).

AUTHOR

Langinfo was developed by HP.

SEE ALSO

date(1), 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 −
LINK_MAX	Max number of links to a single file	1000 +*
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 canoninal 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
reconfig(1M).

Series 800
uxgen(1M).

STANDARDS CONFORMANCE

limits.h: XPG2, XPG3, POSIX.1, FIPS 151-1

NAME

man – macros for formatting entries in this manual

DESCRIPTION

These *nroff* and *troff* macros are used to lay out the format of the entries of this manual. These macros are used by the *man*(1) command.

The default page size is 8.5×11, with a 6.5×10 text area; the **−rs1** option reduces these dimensions to 6×9 and 4.75×8.375, respectively. This option (which is *not* effective in *nroff*) also reduces the default type size from 10-point to 9-point, and the vertical line spacing from 12-point to 10-point. The **−rV2** option may be used to set certain parameters to values appropriate for certain Versatec printers: it sets the line length to 82 characters, the page length to 84 lines, and it inhibits underlining. This option should not be confused with the **−Tvp** option of the *man*(1) command, which is available on some UNIX operating systems.

Any *text* argument below may be one to six "words". Double quotes (**""**) may be used to include blanks in a "word". If *text* is empty, the special treatment is applied to the next line that contains text to be printed. For example, **.I** may 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*.

Type font and size are reset to default values before each paragraph and after processing font- and size-setting macros, e.g., **.I**, **.RB**, **.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*—this corresponds to 0.5 inch in the default page size) by **.TH**, **.P**, and **.RS**, and restored by **.RE**.

.TH *t s c n*	Set the title and entry heading; *t* is the title, *s* is the section number, *c* is extra commentary, e.g., "Optional Software Required", *n* is new manual name (or other text such as "Series 300 Only"). *n* and *c* are concatenated (*c* in parentheses) and placed at the center of the heading line.
.SH *text*	Place subhead *text*, e.g., **SYNOPSIS**, here.
.SS *text*	Place sub-subhead *text*, e.g., **Options**, here.
.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**
.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 subheads (default is 7.2 ens in *troff*, 5 ens in *nroff*).
LL Line length including **IN**.
PD Current interparagraph distance.

Special Features

Starting at HP-UX Release 7.0, the definition of one of the strings used in the page footer macro was moved to the .TH macro and redefined to a non-printing string to prevent the printing of "Hewlett-Packard Company" at the bottom of manual entries that might come from other sources.

This change also 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 but before end of 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**

CAVEATS

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.

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

If any argument to **.TH** contains *any* blanks and is *not* enclosed by double quotes (**" "**), the output might be incorrectly formatted.

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	The maximum value of a single-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: XPG2, XPG3

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 *HP-UX Concepts and Tutorials: Text Formatters.*

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

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 MPE, another HP 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 into MPE by a mere recompilation, and vice versa.

Each routine is described in an individual manual page in section LIBX. They can be grouped in the following categories:

Routines that return language information:

nlgetlang	Return current language.
nlinfo	Return language-dependent information (data tables).
nlnumspec	Return information needed for formatting and converting numbers.

Routines that handle date and time:

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.

Routines that handle language formatted numbers:

nlconvnum	Convert a native language formatted number to an ASCII number.
nlfmtnum	Convert an ASCII number to a language-specific formatted number.

Routines that handle character arrays:

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.

Portnls routines may use flags to select their behavior. The manual pages contain symbolic names for the values of those flags. The 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 logical **and** or by the logical **or** of two of those flags. Logical or/and of two flags is defined as the **or/and** operation performed bit to bit. 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
```

WARNINGS

This library is provided for compatibility with MPE, another HP operating system. Use the Native Language Support routines for C programmers described on *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.

EXTERNAL INFLUENCES

International Code Set Support

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

NAME

rcsintro – description of RCS commands

DESCRIPTION

The 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 later check in commands will ask you for a log entry, which should summarize the changes that you 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 login>
```

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 -r1.3.1 f.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 may be specified in 3 ways:

1) 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.

2) 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"**.

3) 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 will follow a chain of up to ten directory links to reach the RCS directory. Directory links will work with Remote File Access in a LAN environment if you execute an appropriate *netunam* command before executing RCS commands.

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 will replace 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

The 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.

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

The 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 rage 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:

RE RE 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 character of an 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 the ordinary character itself. 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:

RE RE 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 leftmost 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** matches either a string containing **abba** or by a string containing **cde**, but not by a string containing **abbcde** (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.

RE RE 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 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 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 or a bracket expression. For *make*(1) only the part of the pathname following the last slash charater 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 begins with a tilde (~) character, all of the ordinary characters preceding the first slash (or all character 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 may 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), setlocale(3C), environ(5).

NAME

romaji – map of ROMAJI spelling

SYNOPSIS

cat /usr/lib/nlio/romaji

DESCRIPTION

The file **/usr/lib/nlio/romaji** shows how Japanese is spelled using Roman characters. Each line contains three fields separated by a tab character. The first and the second field give the Japanese alphabet in HIRAGANA and KATAKANA respectively. The third field shows how it is spelled using Roman characters.

DEPENDENCIES

NLIO and a KANJI terminal are required in order to display the file **/usr/lib/nlio/romaji**.

FILES

/usr/lib/nlio/romaji

SEE ALSO

RomajiHiragana(3X), RomajiKatakana(3X), RomajiHankakuKatakana(3X)

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)
SIGWINDOW	23	*G*	window change or mouse signal; see windowing package
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 whose parent is the initialization process (see *init*(1M)) stops as the result of receiving the SIGTSTP, SIGTTIN, or SIGTTOU signal, the process terminates because the SIGKILL signal is sent to the stopped process.

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*(2) 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 it was interrupted. The signal interface routines other than *signal*(2) normally do not reset the action for the signal caught. However, *sigaction*(2) and *sigvector*(2) 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 it was interrupted.

When a signal is caught during the execution of system calls such as *read*(2), *write*(2), *open*(2) or *ioctl*(2) on a slow device (such as a terminal, but not a file), during a *pause*(2) system call or a *wait*(2) 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*(2) 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*(2) 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*(2) function, the calling process blocks until all of the child processes of the calling processes terminate. The *wait*(2) 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*(2), *wait3*(2), or *waitpid*(2). Otherwise the presence of the child that caused the original signal causes another signal immediately, resulting in infinite recursion.

When processing a pipeline, the shell makes the last process in the pipeline the parent of the preceding processes. 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

The signal SIGPWR 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* &~ 0x8000000) is the value of the HP98248 status register. Refer to the 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 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 the Series 800. 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)) may be called. Using *longjmp* ensures software portability across different hardware architectures.

Sigaction is not currently supported.

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, XPG3, POSIX.1, FIPS 151-1, ANSI C

NAME

stat – data returned by stat/fstat/lstat system call

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
```

DESCRIPTION

The system calls *stat*, *fstat*, and *lstat* return data whose structure is defined by this include file. The encoding of the field *st_mode* is defined in this file also.

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 */
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 GMT, Jan 1, 1970 */
```

The encoding of the field *st_mode* is defined as follows:

```
#define S_IFMT   0170000 /* type of file */
#define S_IFDIR  0040000 /* directory */
#define S_IFCHR  0020000 /* character special */
#define S_IFBLK  0060000 /* block special */
#define S_IFREG  0100000 /* regular (ordinary) */
#define S_IFIFO  0010000 /* fifo */
#define S_IFLNK  0120000 /* symbolic link */
#define S_ISUID  04000   /* set user id on execution */
#define S_ISGID  02000   /* set group id on execution */
#define S_ISVTX  01000   /* save swapped text even after use */
#define S_IREAD  00400   /* read permission, owner */
#define S_IWRITE 00200   /* write permission, owner */
#define S_IEXEC  00100   /* execute/search permission, owner */
#define S_ENFMT  02000   /* set file-locking mode to enforced */
```

NETWORKING FEATURES

RFA

The contents of the *stat* structure also include the following members:

```
uint    st_remote:1;  /* Set if file is remote */
dev_t   st_netdev;    /* ID of device containing */
                      /* network special file */
ino_t   st_netino;    /* Inode number of network special file */
```

The following additional bit is defined for the field *st_mode*:

```
#define S_IFNWK 0110000 /* network special */
```

FILES

/usr/include/sys/stat.h
/usr/include/sys/types.h

SEE ALSO

stat(2), types(5).

DEPENDENCIES

Series 300 Diskless

The contents of the *stat* structure include the following additional members:

```
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 */
```

The following additional bit is defined for the field *st_mode*:

```
#define S_CDF   04000   /* context-dependent file   */
                        /* if file type is directory */
```

STANDARDS CONFORMANCE

stat.h: SVID2, XPG2, XPG3, POSIX.1, FIPS 151-1

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 RMBASIC. 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.
.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 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/term** 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 which originated from UCB code (such as *vi*(1) and *more*(1)) and commands which 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 it 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

Series 300 Diskless

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

types.h: XPG2, XPG3, POSIX.1, FIPS 151-1

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*(2) 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*(2) and *fcntl*(2) 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 198808L, indicating the (4-digit) year and (2-digit) month that the standard was approved by the IEEE Standards Board.

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*(2) or *pathconf*(2):

_POSIX_CHOWN_RESTRICTED	the use of *chown*(2) is restricted to a process with 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*(2) call (true of all HP-UX implementations)
_POSIX_VDISABLE	terminal special characters can be disabled using this character (see *termio*(7)).

Symbolic constants for *sysconf*(2):

_SC_ARG_MAX
_SC_CHILD_MAX
_SC_CLK_TCK
_SC_JOB_CONTROL

_SC_NGROUPS_MAX
_SC_OPEN_MAX
_SC_PASS_MAX
_SC_SAVED_IDS
_SC_VERSION
_SC_XOPEN_VERSION

Symbolic constants for *pathconf*(2):

_PC_CHOWN_RESTRICTED
_PC_LINK_MAX
_PC_MAX_CANON
_PC_MAX_INPUT
_PC_NAME_MAX
_PC_NO_TRUNC
_PC_PATH_MAX
_PC_PIPE_BUF
_PC_VDISABLE

Symbolic constants for file streams:

STDIN_FILENO	File number of *stdin*.
STDOUT_FILENO	File number of *stdout*.
STDERR_FILENO	File number of *stderr*.

The following are declared as either functions or macros:

_exit()	execlp()	getpgrp()	setgid()
access()	execv()	getpid()	setpgid()
alarm()	execve()	getppid()	setsid()
chdir()	execvp()	getuid()	setuid()
chown()	fork()	isatty()	sleep()
close()	fpathconf()	link()	sysconf()
ctermid()	getcwd()	lseek()	tcgetpgrp()
cuserid()	getegid()	pathconf()	tcsetpgrp()
dup()	geteuid()	pause()	ttyname()
dup2()	getgid()	pipe()	unlink()
execl()	getgroups()	read()	write()
execle()	getlogin()	rmdir()	

SEE ALSO

access(2), chown(2), exit(2), fcntl(2), kill(2), lseek(2), open(2), pathconf(2), sysconf(2), limits(5), termio(7).

AUTHOR

Unistd was developed by Hewlett-Packard Company.

STANDARDS CONFORMANCE

unistd.h: XPG2, XPG3, POSIX.1, FIPS 151-1

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 allows portable procedures that accept variable argument lists to be written. Routines that have variable argument lists (such as *printf*(3S)) but do not use *varargs* are inherently nonportable, as different machines use different argument-passing conventions.

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 will return 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, as 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* (on *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);
}
```

SEE ALSO

exec(2), vprintf(3S).

BUGS

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. *Printf* can tell how many arguments are there by the format.

It is non-portable to specify a second argument of *char*, *short*, or *float* to *va_arg*, since 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: XPG2, XPG3

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 a 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 via the systems 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 manual for your system.

You associate the name you want with a specific device when you create a special file for that device using the *mkdev*(1M) and *mknod*(1M) commands. 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 9 track tape device file names:

/dev/{r}mt/(c#d)#[hml]{c}{n}

where **r** indicates a raw device, **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, and **n** indicates no rewind on close, e.g., **/dev/mt/2mn.**

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

There have been several other naming conventions in the past for similar devices. Using *ln* (on *cp*(1)) to create a link between the old name and the new standard name is useful as a temporary expedient until all the 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 which 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

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 a 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 via the systems 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 manual for your system.

You associate the name you want with a specific device when you create a special file for that device using the *mkdev*(1M) and *mknod*(1M) commands. 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 9 track tape device file names:

/dev/{r}mt/(c#d)#[hml]{c}{n}

where **r** indicates a raw device, **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, and **n** indicates no rewind on close, e.g., **/dev/mt/2mn.**

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

There have been several other naming conventions in the past for similar devices. Using *ln* (on *cp*(1)) to create a link between the old name and the new standard name is useful as a temporary expedient until all the 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 which 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

autochanger – optical autochanger driver

SYNOPSIS

#include <sys/ioctl.h>
#include <sys/ac.h>

ioctl (fildes, ACIOC_WRITE_QUEUE_CONST, ac_queue_const)
int fildes;
struct queue_const *ac_queue_const;

ioctl (fildes, ACIOC_READ_QUEUE_CONST, ac_queue_const)
int fildes;
struct queue_const *ac_queue_const;

ioctl (fildes, ACIOC_READ_QUEUE_STATS, ac_queue_stats)
int fildes;
struct queue_stats *ac_queue_stats;

ioctl (fildes, ACIOC_INITIALIZE_ELEMENT_STATUS)
int fildes;

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.

The autochanger driver orchestrates the swapping of media in and out of the drives so that it appears to the user as if each piece of media has its own drive.

The device files for autochanger media are character special files with a major number of 55 or block special files with a major number of 10.

Minor Numbers

The minor number for autochanger media takes the form 0xScAFSr, with the following designations:

Sc Select code of the mechanical changer
A SCSI address of the mechanical changer
F Reserved for future features (The only valid value is zero.)
Sr Surface for the specified media, encoded as follows:

surface1a	Sr=01
surface1b	Sr=02
surface2a	Sr=03
surface2b	Sr=04
.	.
.	.
.	.
surface32a	Sr=3F
surface32b	Sr=40

The character device file with *Sr*=00 is used for *ioctl*(2). This device is opened before any *ioctl* can be done (see EXAMPLES).

Ioctl Commands

The following *ioctl* commands (see the header file **<sys/ac.h>**) are valid:

ACIOC_WRITE_QUEUE_CONST
This *ioctl* allows the constants that control the queue to be altered.

ACIOC_READ_QUEUE_CONST

This *ioctl* allows the constants that control the queue to be read.

The following structure is passed to *ioctl* for the ACIOC_READ_QUEUE_CONST and ACIOC_WRITE_QUEUE_CONST requests, as defined in header file **<sys/ac.h>**:

```
struct queue_const {
        int wait_time;
        int hog_time;
};
```

The *wait_time* (value in seconds) controls how long the driver waits for another request for the current surface before swapping. The default is 1 second.

The *hog_time* (value in seconds) controls how long a surface can be held in a drive while requests for another surface are pending. The default is 20 seconds.

The *wait_time* cannot exceed the *hog_time*.

ACIOC_READ_QUEUE_STATS

This *ioctl* allows the vital statistics of the queue to be read.

The following structure is passed to *ioctl* for the ACIOC_READ_QUEUE_STATS request, as defined in header file **<sys/ac.h>**:

```
struct queue_stats {
        long size;
        long mix;
};
```

The **size** is the number of requests in the queue.

The **mix** is the number of different surfaces in the queue.

ACIOC_INITIALIZE_ELEMENT_STATUS

This *ioctl* instructs the autochanger to reset its internal medium map. This *ioctl* causes the mechanical changer to look for media in all 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.

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 is true:

[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.

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 <sys/ac.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)"
        error (...);
```

```
/* do the ioctl */
if (ioctl (fildes, ACIOC_READ_QUEUE_CONSTS, & ac_queue_stats) < 0)
        error (...);

/* print the results */
printf ("Queue size is %d\n", ac_queue_stats.size);
printf ("Queue mix is %d\n", ac_queue_stats.mix);
```

DEPENDENCIES

This driver only supports the C1700A Optical Autochanger.

AUTHOR

The optical autochanger driver was developed by HP.

FILES

/dev/ac/*	block special files
/dev/rac/*	character special files

SEE ALSO

mkdev(1M), mknod(1M), ioctl(2).
HP-UX System Administrator Manual.

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 will continue 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 wants 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, and then the data the user has typed into the terminal's memory, followed by a press of the **<ENTER>** key, the terminal will send 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 will send another *trigger* character to the terminal.

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 will be block mode transfers. When block mode transmissions have not been enabled, character mode transmissions will be processed as described in *termio*(7). If block mode transmissions are not enabled, but an *alert* character is received anywhere in the input data, then the transmission mode will be 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 will wait until completion of the data transmission before returning.

The data block terminator character will be 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 will return the requested number of bytes and the remaining bytes will be 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, then 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 will complete a read request when the *alert* character is received. The second *trigger* will be sent by the driver when the application issues the next read.

There are several special characters (both input and output) which 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 has completed. It signifies the end of the data block to the computer.

The two *ioctl*(2) system calls that apply to block mode use the following structure, defined in **<blmodeio.h>**:

```
#define   NBREPLY   64

struct    blmodeio          {
          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[NBREPLY]; /* optional reply */
};
```

The *cb_flags* field 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, then all transfers will be done in raw mode, regardless of the CB_BMTRANS bit. If CB_BMTRANS is not set, then input processing will be 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 will be flushed and the block mode handshake will be invoked. The system will then send the *cb_trig2c* character to the terminal, and a block mode transfer will follow. 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. The block mode handshake is not required and data read is processed as block mode transfer data. The block mode handshake may 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 will be 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 will cause 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 that will be 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* field specifies the length in bytes of the *cb_reply* field. If set to zero, the *cb_reply* string will not be 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 may be included in the reply string. The number of characters sent is specified by *cb_replen*. The initial value of all characters in the *cb_reply* array is null.

On systems that support process group control, unless otherwise noted for a specific *ioctl* command, the *ioctl*s are restricted from use by background processes. An attempt to issue an *ioctl* from a background process will cause the process to block and may cause a SIGTTOU signal to be sent to the process group.

The primary *ioctl*(2) system calls have the form:

ioctl (*fildes, command, arg*)
struct blmodeio **arg;*

The commands using this form are:

CBGETA	Get the parameters associated with the block mode interface and store in the *blmodeio* structure referenced by *arg*. This command 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.

RETURNS

During a read, it is possible for the user's buffer to be altered even if an error value is returned. The data in the user's buffer should be ignored as it will not be complete. The following errors may be returned by various system calls interfacing with this terminal protocol:

Read

[EIO]	An error occurred during the transmission of the block mode data block.

Ioctl

[EINVAL]	An invalid parameter or command value was passed to the *ioctl* call.

WARNINGS

The EIO error that is returned for read errors can be caused by many events. The read will return 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 a fudge factor of ten seconds.

AUTHOR

Blmode was developed by Hewlett-Packard.

SEE ALSO

termio(7).

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 – cartridge tape access

DESCRIPTION

This page describes the actions of the general HP-UX cartridge tape drivers when referring to a 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, like 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 of the tape drive and the media.

Cartridge tape units in either Command-Set 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 HPIB is not blocked for a significant amount of time. For cartridge tape devices that do not support the 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 Command Set 80 (CS/80) disk drives can be accessed as blocked or raw devices. However, using a cartridge tape as a file system will severely limit 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.

DEPENDENCIES

HP7941CT/HP9144A/HP35401

These cartridge tape devices support the immediate report mode.

HP7942/HP7946

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.

HP7908/HP7911/HP7912/HP7914

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

diag0 – diagnostic interface to I/O subsystem

SYNOPSIS

pseudo-device diag0

DESCRIPTION

Diag0 is the diagnostic global port for the system. All diagnostic events are sent to it on subqueue one.

The only commands that *diag0* supports are *open*(2), *close*(2), and *read*(2). The *open*(2)command is exclusive and the opening process must have an effective super-user user ID. A read of the diagnostic device file returns a diagnostic event message. If a diagnostic event has been logged since the last read, the next read call will be satisfied with that event message and will return immediately. If no diagnostic events are queued and waiting, the process that made the read call is put to sleep waiting for an event to be sent to the diagnostic port. When an event finally arrives, the sleeping process is awakened and the read is satisfied. If during the read an error is encountered, the read will be terminated and errno will be set.

RETURNS

Upon successful completion, a zero is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Diag0 does not support any *ioctl*(2) calls or *write*(2) commands. If either command is issued then the following error is returned:

[ENODEV] ioctl not supported.

[EBADF] write not supported.

When opening *diag0* the following errors may be returned:

[EPERM] permission denied, not superuser.

[EBUSY] device file already opened.

When attempting a read of *diag0* the following errors may be returned:

[EIO] error occurred while retrieving message.

[ENXIO] unknown message type received.

AUTHOR

Diag0 was developed by HP.

FILES

/dev/diag0
/hp-ux

SEE ALSO

open(2), close(2), read(2).

NAME

disk – direct disk access

DESCRIPTION

This page describes the actions of the general HP-UX disk drivers when referring to a disk as either a block-special or character-special (raw) device.

Block special files access disks 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 disk records. Each I/O operation results in one or more logical block transactions. The size of the logical block request can be found in f_blksize as returned by *ustat*(2).

There is also a *raw* interface via a character special file which provides for direct transmission between the disk 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 disk 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. Also, each transfer must occur on a sector boundary and must read a whole number of sectors. The sector size is a hardware dependent value.

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.

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 disk is not specified, and in general is unpredictable.

Disk device file names are in the following format:

/dev/{r}dsk/(r)(c#d)#s#

where **r** indicates a raw interface to the disk, the second **r** should not be used and is reserved for future use, the **c#d** indicates the controller number (which is optionally specified by the system administrator), and **#s#** indicates the drive and section numbers, respectively. The assignment of controller, drive, and section numbers is described in the system administrator manual for your system.

WARNING

On some systems, having both a mounted file system and a block special file open on the same device is asking for trouble; this should be avoided if possible. This is because it may be possible for some files to have private buffers in some systems.

DEPENDENCIES

Series 800

All transfers must begin on a DEV_BSIZE boundary. (DEV_BSIZE is defined in <**sys/param.h**>.)

AUTHOR

Disk was developed by HP and AT&T.

SEE ALSO

mkdev(1M), mknod(1M), ct(7), intro(7).

HP-UX System Administrator Manual included with your system.

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.

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 800

When requesting GCMAP, the parameter *arg* is ignored and should be set to **0**.

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

All supported ITEs ignore the frame buffer lock for output.

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/ioctl.h>
#include <sys/gpio.h>

ioctl (fildes, IO_CONTROL, gpio_control)
int fildes;
struct io_ctl_status *gpio_control;

ioctl (fildes, IO_STATUS, gpio_status)
int fildes;
struct io_ctl_status *gpio_status;

ioctl (fildes, IO_ENVIRONMENT, gpio_env)
int fildes;
struct io_environment *gpio_env;
```

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 (filedes, IO_CONTROL, &gpio_control);
```

In the io_ctl_status structure, the *type* field specifies the type of control function required. The *arg* array holds any associated arguments. The defined values for *type* and their use are described 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 **filedes**; thus events can be disabled by using a zero.

Arg[0] defines which events allow the calling process to receive a signal should an asynchronous event occur on the *gpio*. *Arg*[0] contains an event mask which is constructed by computing the bit-by-bit logical OR of the flag below.

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-by-bit logical 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* request of GPIO_GET_CONFIG *type*.

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.

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 (filedes, IO_STATUS, &gpio_status);
```

Type specifies the type of information to return in the *arg* array. The following status requests are allowed 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. *Arg*[0] contains the configuration mask

described above and *arg*[1] contains the handshake delay value in nanoseconds.

GPIO_STS_LINES

This status function gives the user access to the *gpio* status lines. *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 (filedes, IO_ENVIRONMENT, &gpio_env);
```

where the **io_environment** structure includes the following fields:

```
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 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

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* may 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, may be set to enable non-blocking reads (see *open*(2)).

Close(2) returns an HP-HIL keyboard to mapped keycode mode. Its input is now 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 ];
```

The *packet_length* specifies the number of bytes in the packet including itself, and can range from six to twenty bytes. The *time_stamp*, when re-packed into an integer, specifies the time that the system has been running since the last system boot, in tens of milliseconds. The most significant byte of the timestamp is *time_stamp*[0]. The *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, then 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 with one 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. The *ioctl*(2) system calls have the form:

```
int ioctl(fildes, request, arg)
int fildes, request;
char *arg;
```

The following *hil* request codes are defined in <**sys/hilioctl.h**>:

HILID Identify and Describe

This command 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 commands. 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 command.

HILPST Perform Self Test

This command 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 command.

HILRR Read Register

The Read Register command 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 if a device supports the Read Register command.

HILWR Write Register

The Write Register command 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 commands.

HILRN Report Name

The Report Name command 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 command.

HILRS Report Status

The Report Status command returns the device-specific status information string in the character array to which *arg* points. The string may be up to fifteen bytes long. The Extended Describe record indicates support of the Report Status command.

HILED Extended Describe

The Extended Describe command 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 commands. If the device implements the Read Register command, the maximum readable register is specified. If the device supports the Write Register command, 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 command is indicated in the Describe Record Header byte.

HILSC Report Security Code

The Report Security Code command 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 may use this ioctl command 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 command in the Describe Record Header.

HILER1 Enable Auto Repeat Rate = 1/30 Second

This command 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 command does not use *arg*.

HILER2 Enable Auto Repeat Rate = 1/60 Second

This command 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 command does not use *arg*.

HILDKR Disable Keyswitch Auto Repeat

This command turns off the "repeating keys" feature implemented in the firmware of some HP-HIL keyboard and keypad devices. This command does not use *arg*.

HILP1..HILP7 Prompt 1 through Prompt 7

These seven commands 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 commands in the I/O Descriptor Byte in the Describe Record. These commands do not use *arg*.

HILP Prompt (General Purpose)

This command is intended as a general purpose stimulus to the user. Devices accepting this command indicate so in the I/O Descriptor Byte in the Describe Record. This command does not use *arg*.

HILA1..HILA7 Acknowledge 1 through Acknowledge 7

These seven commands 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 command. These commands do not use *arg*.

HILA Acknowledge (General Purpose)

The Acknowledge command is intended to provide an audio or visual response to the user. Devices accepting this command indicate so in the I/O Descriptor Byte in the Describe Record. This command 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) command.

HP-HIL devices have no status bit available to indicate whether they support the HILER1, HILER2, or HILDKR ioctl commands.

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), hilkbd(7), open(2), read(2), select(2), signal(5), 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, PWS, and RMB 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 will lose 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) will return control of the keyboard to the ITE, if present. Any unread input at that time will be discarded.

Read(2) returns data from the keyboard in time-stamped packets:

```
unsigned char time_stamp[4];
unsigned char status;
unsigned char data;
```

The *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) will return **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 may 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 will always return a false indication in the bit masks.

Ioctl(2) is used to perform special operations on the device. The *ioctl*(2) system calls have the form:

```
int ioctl(fildes, request, arg)
int fildes, request;
char *arg;
```

The following *hilkbd* request codes are defined in **<sys/hilioctl.h>**:

KBD_READ_CONFIG
: Read the configuration code.

This command returns in the **char** variable to which *arg* points a one-byte configuration code. 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 command returns in the **char** variable to which *arg* points a one-byte language code, as read from the keyboard. 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 command returns in the **char** variable to which *arg* points a one-byte value containing bit flags specifying the state of the shift and control keys:

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 to 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 <ioctl.h>
#include <sys/hpibio.h>

ioctl (fildes, IO_CONTROL, hpib_control)
int fildes;
struct io_ctl_status *hpib_control;

ioctl (fildes, HPIB_COMMAND, hpib_cmd)
int fildes;
struct hpib_command *hpib_cmd;

ioctl (fildes, IO_STATUS, hpib_status)
int fildes;
struct io_ctl_status *hpib_status;

ioctl (fildes, IO_ENVIRONMENT, hpib_env)
int fildes;
struct io_environment *hpib_env;
```

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*(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	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*(2) and *write*(2) requests are used for data transfer over HP-IB. 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 take 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(s) 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*(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 the *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 type of interface. 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 wishes 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*(2) requests can be performed under what circumstances. The first three columns indicate whether the interface must be in the controlling state given to perform the request. An entry of N means that the interface must not be in that state, while an entry of – means that the state is irrelevant. The next two columns indicate if 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 is true:

[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*(2) 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, an auto-addressed file user 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*(2) or *dup*(2)) that refer to the same device file. 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

HP27110B

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).

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 a 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 via the systems 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 manual for your system.

You associate the name you want with a specific device when you create a special file for that device using the *mkdev*(1M) and *mknod*(1M) commands. 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 9 track tape device file names:

/dev/{r}mt/(c#d)#[hml]{c}{n}

where **r** indicates a raw device, **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, and **n** indicates no rewind on close, e.g., **/dev/mt/2mn**.

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

There have been several other naming conventions in the past for similar devices. Using *ln* (on *cp*(1)) to create a link between the old name and the new standard name is useful as a temporary expedient until all the 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 which 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

iomap – physical address mapping

DESCRIPTION

The *iomap* mechanism allows the mapping (thus direct access) of physical addresses into the user process address space. For Series 300 Models 310 and 320 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 65536. NN*65536 is the size of the region to be mapped. For example, the minor number for a device at 0x720000 and 128k in size is 0x007202.

Access to the iomap devices is controlled by the file permissions set on the character special file.

Multiple processes may concurrently have iomap devices opened and mapped. It is the responsibility of the processes to synchronize their accesses.

Read and *write* system calls are not supported.

Ioctl is used to control the *iomap* device. The valid *ioctl* commands (see <iomap.h>) are:

IOMAPMAP
: map the iomap device into user address space at the location specified in the ioctl argument. If the user address specified in the *ioctl* argument is 0, the system selects an appropriate address. The *ioctl* then returns the user address where the device was mapped, storing it in the original ioctl argument (see EXAMPLES below). Multiple processes may concurrently have the *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 are accessible only through the iomap interface. Consequently, attempts to access them through *shmat*(2), *shmctl*(2), *shmdt*(2), etc. result in EACCESS errors.

WARNING

Iomap devices should be created and used with extreme caution. Inappropriate accesses to io devices or ram may result in a system crash.

ERRORS

[ENINVAL]	Address field out of range, *ioctl* command invalid.
[ENOMEM]	Cannot allocate required memory for mapping.
[ENODEV]	Read/write unsupported.
[ENXIO]	No such address.
[ENOSPC]	Cannot allocate required resources for mapping.
[ENOTTY]	Bad *ioctl* command, or *fildes* is not the file descriptor for an *iomap* device file.

EXAMPLES

Consider the following code fragment:

```
#include <iomap.h>
...
int fildes;
char* addr;
   addr=REQUESTED_ADDRESS;
   ioctl(fildes,IOMAPMAP,&addr);
   printf("actual address=0x%x\n,addr);
```

where **fildes** is the device special file descriptor and **addr** is a pointer to a character variable.

If **addr** is zero, the system selects a suitable address then returns the selected address, in **addr**.

If the value in **addr** is non-zero, 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

lp – line printer

DESCRIPTION

All file names in **/dev** containing the mnemonic *lp* are special files providing the interface to a particular line printer. A line printer is a character special device that may optionally have an interpretation applied to the data.

If the *lp* mnemonic is preceded by the character **r**, data is sent to the printer in *raw mode*. (This could assume, for example, a graphic printer operation.) In raw mode, no interpretation is done on the data to be printed, and no page formatting is performed. The bytes are simply sent to the printer and printed as is.

If the *lp* mnemonic is not preceded by the character **r**, data can be sent to the printer in *raw mode*. A new feature has been developed to allow printers to receive data for both graphics and page printer operations. In raw mode, no interpretation is done on the data to be printed, and no page formatting is performed. The bytes are simply sent to the printer and printed as is. Raw mode is set and cleared by the ioctl command LPRSET. The new feature applies *only* to mneumonics not preceded by the character **r**.

If the *lp* mnemonic is not preceded by the character **r**, the 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) system calls 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.

```
#include <sys/lprio.h>
ioctl (fildes, command, arg)
struct lprio *arg;
```

The *commands* are:

LPRGET Get the current printer status information and store in the **lprio** structure referenced by **arg**.

LPRSET Set the current printer status information from the structure referenced by **arg**.

These are remembered across opens, and thus, indent, page width and page length can be set with an external program. If the columns field is set to zero, the defaults are restored at the next open.

The structure passed to the LPRGET and LPRSET *ioctl* calls, as defined in **<sys/lprio.h>**, is:

```
struct lprio {
        short   ind;        /* indent */
        short   col;        /* columns per page */
        short   line;       /* lines per page */
        short   bksp;       /* backspace handling flag */
        short   open_ej;    /* pages to eject on open */
        short   close_ej;   /* pages to eject on close */
        short   raw_mode;   /* raw mode flag */
};
```

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 will be 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 will not be 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 will be formatted according to the prior format settings.

DEPENDENCIES

Series 300

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 Manual* 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), 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.

The 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

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 will be considered to be PENDING if it is waiting for the STATUS to be raised. An *open* to a port will be 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:

ioctl (fildes, command, arg)
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:

ioctl (fildes, command, arg)
struct 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 page describes the behavior of HP magnetic tape interfaces and controls. The files **/dev/mt/*** and **/dev/rmt/*** refer to specific tape drives; the behavior of the specific unit is specified in the major and minor numbers of the device special file that describes the tape unit.

The following naming convention is recommended for tape devices because it connects most of the mode flags with the device name:

/dev/{r}mt/(c#d)#[hml]{c}{n}

In this format, **r** indicates a raw device, **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, and **n** indicates no rewind on close. For example, **/dev/rmt/2mn** is raw device 2 at 1600 bpi with no rewind and no compression. Blocked magnetic tapes are used only for special situations and are supported only in some implementations. See DEPENDENCIES below for details. The selection of controller and unit numbers is system dependent, and is discussed in the appropriate system administrator's manual.

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 on raw tapes only. 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.

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.

The special files associated with a raw tape interface are named **rmt/***. Unless otherwise stated, the following discussion refers to raw magnetic tapes.

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.

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.

Spacing operations (back or forward space file or record) leave the tape positioned past the object being spaced to in the direction of motion. That is, 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 raw magnetic tape device are ignored. Instead, the *ioctl*(2) operations below can be used to position the tape and determine its status.

The following is included from <**sys/mtio.h**> and describes the possible 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 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 */

/* 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;

/* The following are used only when block devices are supported */
```

```
        daddr_t    mt_fileno;
        daddr_t    mt_blkno;
};
/*
* Constants for mt_type; the first three are historical
*/
#define MT_ISTS          01
#define MT_ISHT          02
#define MT_ISTM          03
#define MT_IS7970E       04
#define MT_ISSTREAM      05
```

WARNINGS

It is impossible to write a program that leaves a tape positioned at the beginning of the tape on an AT&T–style raw 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.

DEPENDENCIES

Series 300

Block magnetic tape is not supported.

Series 800

The MTNOP operation does not set the device independent status word.

The files **/dev/mt/*** refer to block magnetic tapes. They should be used only for system installation or for treating a previously written magnetic tape as a read-only block file system.

A read-only block tape should be created with *dd*(1) using raw mode and a record size of 512 or 1024 bytes. (The default record size is 512 bytes.) The driver requires exactly 512-byte records on the physical tape, or exactly 1024-byte records if hex 0x8000 is OR'ed into the minor number. The 1024-byte option is intended for use with HP INSTALL/SUPPORT type file systems.

Although the size of records on a block tape is always 512 (or 1024) bytes, the block I/O system deals with block sizes as a multiple of DEV_BSIZE (defined in <**sys/param.h**>), with the tape driver making the translation. For this reason, if a user writes 512 bytes (for example) in block mode, the block I/O system attempts to read from the block tape prior to incorporating the new data and writing it to the tape. Since reading a blank tape or a tape of unknown format terminates in an error, it is advised that *dd*(1) be used to create the tape as described above, and that block magnetic tape be used only for seeking and reading. Alternatively, always write a byte count that is a multiple of BLKDEV_IOSIZE (defined in <**sys/param.h**>) bytes to avoid the hazards of an erroneous prior block read.

A tape treated as a block-special device consists of several 512-byte records terminated by an EOF.

The system enables a previously written block tape to be treated as an ordinary file, except that writing in the middle of a file truncates the file at that point. Seeks have their usual meaning and it is possible to read or write a byte at a time.

The 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_EOF(x)          /* At an EOF mark */
GMT_BOT(x)          /* At beginning of tape */
GMT_EOT(x)          /* At end of tape */
GMT_WR_PROT(x)      /* Tape is write protected */
GMT_ONLINE(x)       /* Drive is online */
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_DR_OPEN(x)      /* Drive door is open */
GMT_IM_REP_EN(x)    /* Immediate reporting mode enabled */
```

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/mt/*
/dev/rmt/*

SEE ALSO

dd(1), mt(1), ioctl(2), ct(7).

NAME

nlio – Native Language I/O server

DESCRIPTION

The Native Language I/O server runs between the terminal device of an actual terminal and the master side of a pseudoterminal. The Native Language I/O interface is available on the slave side of the pseudoterminal. (See *pty*(7) for information on pseudoterminals.) All I/O to the terminal is performed through the server.

Several *ioctl*(2) requests are provided to control Native Language I/O on the terminal. The *ioctl*(2) system call has the form:

```
#include <sys/ioctl.h>
#include <nlio.h>

int ioctl(fildes, request, arg)
int fildes, request;
char arg[NLIOREQLEN];
```

The include file <**sys/ioctl.h**> must be included before <**nlio.h**>.

The buffer specified by *arg* must not be less than **NLIOREQLEN** bytes in length.

Request can have the following values:

ISNLIO Check whether a Native Language I/O server is running for the device associated with *fildes*. The buffer specified by *arg* is not used. If the server is running, **NLIOSERV** is returned; otherwise, **-1** is returned.

NLIOGSERV Get the name of the server that is running for the terminal. The name is returned to the buffer specified by *arg*. The name is terminated with '\0'.

NLIOGTB Get the current terminal behavior. The name of terminal behavior is returned to the buffer specified by *arg*. It is terminated with '\0'. If no terminal behavior is currently specified, only '\0' is returned. This can occur if an unknown terminal behavior was previously specified.

NLIOSTB Set terminal behavior. The current terminal behavior is changed to the terminal behavior in the buffer specified by *arg*. It must be terminated with '\0'.

NLIOGFLAGS Get the current values of the flags. The byte flags are returned in the buffer specified by *arg*. If a particular flag is not available for the device, '\0' is returned in that flag's buffer position. Only the following flag is defined at this time:

arg[**NLIO**]
This flag controls 16-bit data parsing and conversion, and has one of these values:
NLIO_I enable input parsing only
NLIO_O enable output conversion only
NLIO_IO enable both input parsing and output conversion
NLIO_N enable neither of the above

NLIOSFLAGS Set flags. See NLIOGFLAGS for description of the flags. The current values of the flags are changed to the values passed in the buffer specified by *arg*. If a flag value is '\0', it is ignored and that flag's current value is not changed. Clear the whole array of **NLIOREQLEN** flags with '\0' characters before setting the new values.

RETURN VALUE

A value of **NLIOSERV** is returned if the Native Language I/O server is running for the device associated with *fildes* and no errors have occurred.

If *ioctl*(2) is called with invalid arguments, a value of **−1** is returned and **errno** is set as described in *ioctl*(2).

If a request error has occurred, a value of **−1** is returned and **errno** is set to EINVAL.

ERRORS

After a request error has occurred, the NLIOGERR request is available to determine what is wrong. The form of this call is:

```
#include <sys/ioctl.h>
#include <nlio.h>

int ioctl(fildes, request, error)
int fildes, request;
int *error;
```

NLIOGERR Get the error code of the last unsuccessful request. The error code is returned in the integer variable pointed to by *error*, and will be one of the following:

UNKNOWN_TB	An unknown terminal behavior was passed.
BAD_TB	The terminal behavior requested was invalid or not appropriate.
INVALID_FLAG	The flags supplied with NLIOSFLAGS were invalid or unknown.
TOO_LONG_NAME	The specified name was too long.
UNKNOWN_REQ	An unknown *nlio* request was passed.

WARNINGS

Do not set the tty special control characters such as INTR, ERASE, and KILL (see *termio*(7)) to ASCII characters with values in the range of **'\021'** to **'\0176'** inclusive for the terminal with which the Native Language I/O server is associated, because HP-15 can include these values as the second byte of a 16-bit character code.

Clear the whole array of **NLIOREQLEN** bytes with **'\0'** characters before setting flags with NLIOSFLAGS, even if only one flag is used. Otherwise, unexpected things can happen to the I/O.

AUTHOR

Nlio was developed by HP.

SEE ALSO

nlio(1), nlioenv(1), nliostart(1), nlioinit(1M), ioctl(2), pty(7), termio(7).

EXTERNAL INFLUENCES

International Code Set Support

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

NAME

null – null file

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

STANDARDS CONFORMANCE

null: SVID2, XPG2, XPG3

NAME

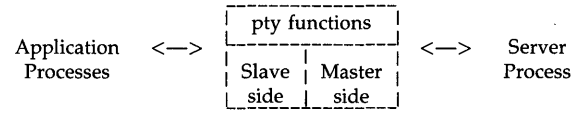
pty – pseudo terminal driver

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). However, whereas all other devices that provide the interface described in *termio*(7) have a hardware device of some sort behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, 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. 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*(2) or *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*(2) request made on the slave side of a pty after the master side is closed returns 0 bytes. 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.

Processing *ioctl*(2) requests

By default, any *ioctl*(2) request defined by *termio*(7) is recognized by both the master and slave sides of a pty. These *ioctl*(2) requests are processed by the pty driver as specified by *termio*(7). In addition, the *ioctl*(2) requests defined below are recognized by the master side of a pty. The slave side only recognizes *termio*(7) *ioctl*(2) requests. An *ioctl*(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. An *ioctl*(2) request not recognized by the pty returns **–1** and sets the external variable **errno** to EINVAL. Note, some of the master-side-only *ioctl*(2) requests affect which *ioctl*(2) requests are recognized by the master and slave side of the pty. These master-side-only *ioctl*(2) requests also affect the way recognized *ioctl*(2) requests, *open*(2) requests, and *close*(2) requests are processed by the pty driver.

The following *ioctl*(2) 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*(7) processing by a pty. *Termio*(7) processing is enabled if the **int** addressed by *arg* is nonzero and disabled if the **int** addressed by *arg* is zero. By default, *termio*(7) processing is enabled. *Termio*(7) processing refers to processing of input and output described by *termio*(7) (such as tab expansion), as well as the processing of the *ioctl*(2) requests described by *termio*(7). When disabled, all input and output data is passed through the pty without modification. Issuing a TIOCTTY *ioctl*(2) 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*(2) requests: TIOCPKT, TIOCREMOTE, TIOCBREAK, TIOCSTOP, TIOCSTART, TIOCTRAP, and TIOCMONITOR.

When TIOCTTY is enabled, all *termio*(7) *ioctl*(2) requests execute as specified in *termio*(7), regardless of the side from which the *ioctl*(2) request is made. When TIOCTTY is disabled, master side *termio*(7) *ioctl*(2) requests set and return the the external variable **errno** to EINVAL. Slave side *termio*(7) *ioctl*(2) requests are processed like any other *ioctl*(2) request when TIOCTTY is disabled. In particular, slave side *termio*(7) *ioctl*(2) requests set and return the external variable **errno** to EINVAL when both TIOCTTY and TIOCTRAP are disabled. (See the discussion of *ioctl*(2), *open*(2), *close*(2) trapping below). *Ioctl*(2) 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*(2) requests, defined in **<sys/ptyio.h>**, apply only to the master side of a pty. In particular, these *ioctl*(2) requests enable/disable specific modes of pty driver operation. These *ioctl*(2) requests work in series with TIOCTTY; that is, the mode must be enabled by its *ioctl*(2) 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*(2) 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/^Q.

TIOCPKT_NOSTOP
Stop and start characters are set to something other than ^S/^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 like 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*(2) request flushes all data buffered in the pseudo terminal.

The following *ioctl*(2) requests, defined in <**sys/ptyio.h**>, apply only to the master side of pty. In particular, these *ioctl*(2) requests are only recognized when TIOCTTY is enabled. When TIOCTTY is disabled, these *ioctl*(2) 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 though a user had hit 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 (that is, like typing ^S). Takes no parameter.

TIOCSTART Restart output (stopped by TIOCSTOP or by typing ^S). Takes no parameter.

Flow Control *input, 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*(2) from the master side of a pty will return a STOP character. When INPUT is restarted, the next *read*(2) from the master side will return a START character. If packet mode (TIOCPKT) is enabled, the STOP or START character will be preceded by a data packet indicator (TIOCPKTDATA). *Select*(2) should be used by the master-side server before each *write*(2) request to properly handle INPUT flow control. Otherwise, data being written to the master side of a pty may be lost.

When packet mode (TIOCPKT) is disabled and OUTPUT is stopped (see IXON, input modes in *termio(7)*), each subsequent *read*(2) from the master side of a pty will return with no data read. When OUTPUT is restarted, each subsequent *read*(2) from the master side returns data written on the slave side. If packet mode (TIOCPKT) is enabled, the first *read*(2) after OUTPUT has been stopped will return a TIOCPKTSTOP packet. All subsequent reads from the master side while OUTPUT is stopped will return a TIOCPKTDATA packet with no data. When OUTPUT is restarted, the next *read(2)* from the master side will return a TIOCPKTSTART packet. All subsequent reads from the master side will return data written on the slave side preceded by a

TIOCPKTDATA packet. *Select*(2) should be used by the master-side server before each *read*(2) 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*(2), *open*(2), *close*(2) Requests

When trapping is enabled, the master side is notified when the application on its slave side makes an *ioctl*(2), *open*(2), or *close*(2) request. For trapped *ioctl*(2) and *open*(2) 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*(2) requests, the slave slave does not block for an acknowledgement.

Select(2) should be used by the master side server to receive notification of trapped *ioctl*(2), *open*(2), and *close*(2) requests. When one of these requests is trapped, the *select*(2) 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*(2) is not available.

When trapping is disabled (the default), *ioctl*(2) requests made to the slave side set and return the external variable **errno** to EINVAL if they are not recognized by the slave side of the pty driver. The only *ioctl*(2) requests recognized by the slave side are those defined by *termio*(7) and only when TIOCTTY is enabled. When TIOCTTY is disabled, no *ioctl*(2) 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*(2) 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*(2) and *close*(2) requests is enabled or disabled by TIOCTRAP. Trapping *ioctl*(2) requests not defined by *termio*(7) are enabled or disabled by TIOCTRAP. Trapping *ioctl*(2) requests defined by *termio*(7) are enabled or disabled by TIOCTRAP only when TIOCTTY is also disabled. When TIOCTTY is enabled, trapping *ioctl*(2) 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*(2) request.

The following *ioctl*(2) calls apply only to the master side of a pty and pertain to trapping *ioctl*(2), *open*(2), and *close*(2) requests. They are defined in **<sys/ptyio.h>**:.

TIOCTRAP
: Enable or disable trapping of *ioctl*(2), *open*(2), and *close*(2) 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*(2), *open*(2), or *close*(2) 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*(2) "exceptional condition" is not available.

TIOCREQCHECK
: Return the trapped *ioctl*(2), *open*(2), or *close*(2) information to the master side. Use TIOCREQCHECK in response to either a *select*(2) "exceptional condition" or a TIOCTRAPSTATUS indicating that a trap is pending. A TIOCREQCHECK reads the pending *ioctl*(2), *open*(2), or *close*(2) 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*(2) command received.

argget The *ioctl*(2) request applied to master side to receive the trapped *ioctl*(2) 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*(2) request applied to master side to send back the resulting *ioctl*(2) 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*(2) on the slave side.

return_value The success value (initialized to zero) returned by *ioctl*(2) on the slave side when **errno_error** is not set.

When the *ioctl*(2) argument received on the slave side is not a pointer, its value is stored as four bytes retrievable with an *ioctl*(2) request to the master side equal to **argget**.

When an *open*(2) or *close*(2) 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*(2) structure. When TIOCTTY is enabled, the *termio*(7) definition of open/close is executed first, before being passed to the master side. Note, 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*(2), *open*(2), or *close*(2) trap is pending. Accordingly, a TIOCREQCHECK that returns EINVAL in response to a *select*(2) "exceptional condition" indicates that the trapped *ioctl*(2), *open*(2), or *close*(2) request was terminated by a signal after *select*(2) returned.

TIOCREQGET Identical to TIOCREQCHECK except when no *ioctl*(2), *open*(2), or *close*(2) trap is pending. A TIOCREQGET blocks until a slave side *ioctl*(2), *open*(2), or *close*(2) 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*(2) 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*(2) 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*(2). Setting either **return_value** or **errno_error** for a trapped *open*(2) or *close*(2) 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*(2), *open*(2), or *close*(2) 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*(7) *ioctl*(2) 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*(7) *ioctl*(2) requests to be trapped by TIOCMONITOR. TIOCMONITOR trapping can be enabled or disabled regardless of the state of TIOCTTY.

When TIOCTTY is disabled, *termio*(7) *ioctl*(2) requests are not trapped by TIOCMONITOR. However, *ioctl*(2) 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*(2) 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*(7) requests trapped by TIOCMONITOR is the same as the mechanism described above for requests trapped by TIOCTRAP. (It is recommended that *termio*(7) *ioctl*(2) 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*(7) *ioctl*(2) requests are not trapped.)

When using *select*(2) on the master side of a pty, the "exceptional condition" refers to an *open*(2), *close*(2), or *ioctl*(2) request pending on the slave side, while "ready for reading or writing" refers to a *read*(2) or *write*(2) request pending.

Of the *ioctl*(2) 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*(7) *ioctl*(2) request to the slave side, all other *ioctl*(2) requests to the same pty slave side are blocked until the first one is handshaked back by the master side. (*Ioctl*(7) requests that are not trapped, such as *termio*(7) when TIOCTTY is enabled and TIOCMONITOR is disabled, are not blocked.) This permits the implementation of indivisible operations by an *ioctl*(2) call on the slave side that is passed to

the server process.

In summary, the following method of handling trapped *ioctl*(2), *open*(2), and *close*(2) requests is preferred:

Call *select*(2).
This system call blocks the master side until a slave side *ioctl*(2), *open*(2), or *close*(2) request is trapped.

Make TIOCREQCHECK *ioctl*(2) request.
This step returns information about a trapped *ioctl*(2) *open*(2) or *close*(2) request. If TIOCREQCHECK returnsthe external variable **errno** error EINVAL, loop back to the *select*(2) call.

Make **argget** *ioctl*(2) request.
This optional step is used if **argget** is nonzero and the server wants to do more than just reject the trapped slave *ioctl*(2) request.

Make **argset** *ioctl*(2) request.
This optional step is done if **argset** is nonzero and the server wants to pass back a modified *ioctl*(2) structure. It is done after the trapped *ioctl*(2) request is processed via the server on the master side.

Set **errno_error** and **return_error**.
If the trapped request is an *ioctl*(2), set **errno_error** appropriately. If the appropriate value for **errno_error** is zero, **return_error** must be set.

Make TIOCREQSET *ioctl*(2) request.
This step completes the trapped *ioctl*(2), *open*(2), *close*(2) 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*(2) request allows the server process to control how the pty responds when a signal attempts to interrupt a trapped *open*(2) or *ioctl*(2) 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*(2) or *ioctl*(2) 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*(2) 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*(2) continues to be blocked.

TIOCSIGABORT
Prevent a trapped *open*(2) or *ioctl*(2) 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*(2) or *ioctl*(2) 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 be 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*(2) 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

The largest *ioctl*(2) argument passable between master and slave sides is currently limited to 128 bytes.

Slave-side non-*termio*(7) *ioctl*(2) requests either go unrecognized or are passed to the master side, depending upon the state of the TIOCTRAP.

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

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), since 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 page.

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

termio – 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*(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*(1M)) and become a user's standard input, output, and 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*(2) or *tcgetpgrp*(2)). 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*(2). A process relinquishes its controlling terminal if it creates a new session with *setsid*(2) or *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*(2) 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*(2) 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:

1. If there is enough data available to satisfy the entire request, *read*(2) completes successfully, having read all of the data requested, and returns the number of characters read.
2. If there is not enough data available to satisfy the entire request, *read*(2) completes successfully, having read as much data as possible, and returns the number of characters read.
3. If there is no data available, *read*(2) 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*(2) 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*(2) 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*(2) 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*(2) 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*(2) 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*(2), 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, like 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. Typically command interpreter processes 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 not 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 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 done.

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*(2) 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 <**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 <**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* (see **termio Caveats**) structures. Any of the bits in the modes fields which 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, then parity generation is enabled but input parity checking is disabled and 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 interelationship between the flags IGNBRK, BRKINT, IGNPAR, and PARMRK. The column marked **Input** gives various types of input characters received. A **0** indicates a NUL character ('\0'), **C** indicates a character other than NUL, **P** indicates a detected parity error, and **F** indicates a detected framing error. Items enclosed in brackets indicate one or more of the conditions are true. If the INPCK flag is clear, characters received with parity errors

will not be 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 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 do just 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, which 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:
B0	Hang up
B50	50 baud
B75	75 baud
B110	110 baud
B134	134.5 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B900	900 baud
B1200	1200 baud
B1800	1800 baud
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
CSIZE	Character size:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits
CSTOPB	Send two stop bits, else one.
CREAD	Enable receiver.
PARENB	Parity enable.
PARODD	Odd parity, else even.
HUPCL	Hang up on last close.
CLOCAL	Local line, else dial-up.
LOBLK	Reserved for use by *shl*(1).

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 stops 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*(2) 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*(2) and *write*(2) 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, 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:

for:	*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*(2) 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

The 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	Description
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*(3C) function is successfully called. Refer to the appropriate manual pages for details.

termios Structure Functions

Function	Description
cfgetospeed(3C)	get output baud rate
cfgetispeed(3C)	get input baud rate
cfsetospeed(3C)	set output baud rate
cfsetispeed(3C)	set input baud rate
tcgetattr(3C)	get terminal state
tcsetattr(3C)	set terminal state

termio Structure Related IOCTL Commands

Several *ioctl*(2) system calls apply to terminal files that use the *termio* structure (see **termio Structure**). If a command is requested which is not recognized, the request returns −1 with **errno** set to EINVAL.

Ioctl(2) 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 *termio* 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 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 pages for details.

Structure Independent Functions

Function	Description
tcsendbreak(3C)	send a break
tcdrain(3C)	wait until output has drained
tcflush(3C)	flush input or output queue or both
tcflow(3C)	suspend or resume input or output
tcgetpgrp(3C)	get foreground process group id
tcsetpgrp(3C)	set foreground process group id

System Asynchronous I/O IOCTL Commands

The following *ioctl*(2) 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 which 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. The super-user can designate that any process receive the SIGIO signals. If the request is not made by the super-user and the calling process does not either designate that itself or another process whose real or saved 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*(2) 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*(3C) function performs the same function.

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*(3C) function performs the same functions.

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*(3C) function performs the same functions.

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*(2) 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:

1. 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;
2. 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;
3. 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:

1. 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;
2. 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;
3. 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 *tcgetattr*(2), *tcsetattr*(2), *tcgetpgrp*(2), and *tcsetpgrp*(2) functions.

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 is used to 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 functions the same as **t_suspc**, except that the *suspend* signal (SIGTSTP) is sent when a process reads the character, rather than when it is typed. When enabled, the typical value for this character is Control-Y or ASCII EM.

Attempts to set any of the reserved characters to a value other than _POSIX_VDISABLE cause the *ioctl* to return −1 with **errno** set to EINVAL with no change in value of the reserved character.

Ioctl(2) 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*(2) 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*(3C) function.

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*(3C) function.

If the *octl* 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**).

WARNINGS

Because various HP-UX implementations use non-serial interfaces that look like terminals (such as internal CRTs) or "smart cards," which cannot implement the exact capabilities described above, not all the 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 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 command interpreter processes. FIOSSAIOOWN is similar to 4.2 BSD FIOSETOWN, with additional 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 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

Data loss can occur with HP 98626/98644 serial interfaces 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).

The *c_iflag* field parameter IXANY (enable any character to restart output) is not supported by the HP 98628B interface card.

Timed delays are not supported.

The HP 98628B interface does not support the following baud rates: 900, 7200, 38400.

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.

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).

STANDARDS CONFORMANCE

termio: SVID2, XPG2

termios: XPG3, POSIX.1, FIPS 151-1

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 wish 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: SVID2, XPG2, XPG3

Section 9: Glossary

NAME

intro – introduction to glossary section

DESCRIPTION

This section contains a glossary of common HP-UX terms. References to other HP-UX documentation are included as appropriate. References to entities such as *wait*(2), *sh*(1), or *fopen*(3S) refer to entries in the other sections of this manual. References to items in italics but having no parenthetical suffixes refer to other entries in this glossary. Any italicized manual names refer to separate manuals that are either included with your system or available separately.

The definitions specifically reflect the HP-UX operating system, although some terms and definitions are also derived from those in the emerging IEEE POSIX standards and the *X/Open Portability Guide*. Differences in wording exist to more specifically reflect characteristics of the HP-UX system.

SEE ALSO

The introduction to this manual.

The glossary section of the *Documentation and Terminology Guide*.

.o ("dot-oh") The suffix customarily given to a relocatable object file. The term, ".o file," is sometimes used to refer to a relocatable object file; the format of such files is sometimes called ".o format." See *a.out*(4).

a.out The name customarily given to an executable object code file on HP-UX. The format is machine-dependent, and is described in *a.out*(4) for each implementation. Object code that is not yet linked has the same format, but is referred to as a *.o* ("dot-oh") file. *A.out* is also the default output file name used by the linker, *ld*(1).

absolute path name
A path name beginning with a slash (/). It indicates that the file's location is given relative to the root directory (/), and that the search begins there.

access Access concerns the process of obtaining data from or placing data in storage, or the right to use system resources. Accessibility is governed by three process characteristics: the effective user ID, the effective group ID, and the group access list. The *access*(2) system call determines accessibility of a file according to the bit pattern contained in its *amode* parameter, which is constructed to read, write, execute or check the existence of a file. The *access*(2) system call uses the *real user ID* instead of the *effective user ID* and the *real group ID* instead of the *effective group ID*.

access groups The group access list is a set of *supplementary group IDs* used in determining resource accessibility. Access checks are performed as described below in *file access permissions*.

access mode An access mode is a form of access permitted to a file. Each implementation provides separate read, write, and execute/search access modes.

address A number used in information storage or retrieval to specify and identify memory location. An *address* is used to mark, direct, indicate destination, instruct or otherwise communicate with computer elements.

In mail, *address* is a data structure whose format can be recognized by all elements involved in transmitting information. On a local system, this might be as simple as the user's *login* name, while in a networked system, *address* specifies the location of the resource to the network software.

In a text editor (such as *ed*), an *address* locates the line in a file on which a given instruction is intended.

For *adb*, the *address* specifies at what assembly-language instruction to execute a given command.

In disk utilities such as *fsdb*, *address* might refer to a raw or *block special file*,*the inode* number, *volume header*, or other file attribute.

In the context of peripheral devices, *address* refers to a set of values that specify the location of an I/O device to the computer. The exact details of the formation of an address differ between systems. On the Series 300, the address is composed of up to four elements: the *select code*, *bus address*, *unit number (ID)*, and *volume number (ID)*.

address space The range of memory locations to which a process can refer.

affiliation See *terminal affiliation*.

appropriate privileges
Each implementation provides a means of associating privileges with a process, for function calls and function call options requiring special privileges. In HP-

UX, *appropriate privileges* refers either to superuser status or to a privilege associated with privilege groups (see *setprivgrp*(1M)).

archive A file comprised of the contents of other files, such as a group of object files (that is, *.o*) used by the linker, *ld*(1)). An archive file is created and maintained by *ar*(1) or similar programs, such as *tar*(1) or *cpio*(1). An *archive* is often called a *library*.

ASCII An acronym for American Standard Code for Information Interchange. ASCII is the traditional System V coded character set and defines 128 characters, including both control characters and graphic characters, each of which is represented by 7-bit binary values ranging from 0 through 127 decimal.

background process group
Any process group that is a member of a session which has established a connection with a controlling terminal that is not in the foreground process group.

backup The process of making a copy of all or part of the file system in order to preserve it, in case a system crash occurs (usually due to a power failure, hardware error, etc.). This is a highly recommended practice.

block (1) The fundamental unit of information HP-UX uses for access and storage allocation on a mass storage medium. The size of a block varies between implementations and between file systems. In order to present a more uniform interface to the user, most system calls and utilities use *block* to mean 512 bytes, independent of the actual block size of the medium. This is the meaning of *block* unless otherwise specified in the manual entry.

(2) On media such as 9-track tape that write variable length strings of data, the size of those strings. *Block* is often used to distinguish from *record;* a block contains several records, whereas the number of records denotes the blocking factor.

block special file A special file associated with a mass storage device (such as a hard disk or tape cartridge drive) that transfers data in multiple-byte blocks, rather than by series of individual bytes (see *character special file*). *Block special files* can be mounted. A *block special file* provides access to the device where hardware characteristics of the device are not visible.

boot or *boot-up* The process of loading, initializing, and running an operating system.

boot area A portion of a mass storage medium on which the volume header and a "bootstrap" program used in booting the operating system reside. The *boot area* is reserved exclusively for use by HP-UX.

boot ROM A program residing in ROM (Read-Only Memory) that executes each time the computer is powered up and is designed to bring the computer to a desired state by means of its own action. The first few instructions of a bootstrap program are sufficient to bring the remainder of the program into the computer from an input device and initiate functions necessary for computation. The function of the boot ROM is to run tests on the computer's hardware, find all devices accessible through the computer, and then load either a specified operating system or the first operating system found according to a specific search algorithm.

bus address A number which makes up part of the address HP-UX uses to locate a particular device. The *bus address* is determined by a switch setting on a peripheral device which allows the computer to distinguish between two devices connected to the same interface. A *bus address* is sometimes called a "device

address".

character An element used for the organization, control, or representation of text. Characters include *graphic characters* and *control characters.*

character set A set of characters used to communicate in a native or computer language.

character special file
A special file associated with I/O devices that transfer data byte-by-byte. Other byte-mode I/O devices include printers, nine-track magnetic tape drives, and disk drives when accessed in "raw" mode (see *raw disk*). A *character special file* has no predefined structure.

child process A new process created by a pre-existing process via the *fork*(2) system call. The new process is thereafter known to the pre-existing process as its *child process*. The pre-existing process is the *parent process* of the new process. See *parent process* and *fork*.

clock tick
A rate used within the system for scheduling and accounting. It consists of the number of intervals per second as defined by {CLK_TCK} that is used to express the value in type **clock_t**. {CLK_TCK} was previously known as the defined constant HZ.

cluster A set of one or more systems, connected by a network, that share a single *file hierarchy*.

This term applies to implementations that include the Series 300 Diskless Package.

cluster node (cnode)
One of the nodes, or systems, in a *cluster*.

This term applies to implementations that include the Series 300 Diskless Package.

cluster server process (CSP)
A special kernel process that runs on a machine in a *cluster* to satisfy requests from other nodes in the cluster. There are two kinds of CSP, the "limited CSP" (LCSP) and the "general CSP" (GCSP). Each node in a cluster runs exactly one LCSP that is necessary to maintain the cluster itself. In addition, the *root server* runs some number of GCSPs, which satisfy user requests for remotely available resources such as files and swap space.

This term applies to implementations that include the Series 300 Diskless Package.

cnode ID A numeric value used to identify each node in a *cluster*. Each *cnode ID* is unique within the cluster.

This term applies to implementations that include the Series 300 Diskless Package.

cnode name A character string used to identify each node in a *cluster*. Each *cnode name* is unique within the cluster.

This term applies to implementations that include the Series 300 Diskless Package.

coded character set
A set of unambiguous rules that establishes a character set and the one-to-one relationship between each character of the set and its corresponding bit

representation. *ASCII* is a *coded character set.*

collating element
The smallest entity used in collation to determine the logical ordering of strings (that is, the *collation sequence*). To accomodate native languages, a collating element consists of either a single character, or two or more characters collating as a single entity. The current value of the *LANG* environment variable determines the current set of collating elements.

collation
The logical ordering of strings in a predefined sequence according to rules established by precedence. These rules identify a collation sequence among the collating elements and also govern the ordering of strings consisting of multiple collating elements, to accomodate native languages.

collation sequence
The ordering sequence applied to *collating elements* when they are sorted. To accomodate native languages, *collation sequence* can be thought of as the relative order of *collating elements* as set by the current value of the *LANG* environment variable. Characters can be omitted from the collation sequence, or two or more collating elements can be given the same relative order (see *string*(3C)).

command
A directive to perform a particular task. In HP-UX, commands are executed through a *command interpreter* called a *shell*. HP-UX supports several shells, including *sh*(1), *csh*(1), *ksh*(1), and *pam*(1). Most commands are carried out by an executable file, called a *utility*, which might take the form of a stand-alone unit of executable object code (a program) or a file containing a list of other programs to execute in a given order (that is, a shell script). Scripts can contain references to other scripts, as well as to object-code programs. A typical *command* consists of the utility name followed by arguments that are passed to the utility. For example, in the command, "ls mydirectory," "ls" is the utility name and "mydirectory" is an argument passed to the "ls" utility.

command interpreter
A program which reads lines of text from standard input (typed at the keyboard or read from a file), and interprets them as requests to execute other programs. A command interpreter for HP-UX is called a *shell*. See *sh*(1), *csh*(1), *ksh*(1), and *pam*(1).

composite graphic symbol
A graphic symbol consisting of a combination of two or more other graphic symbols in a single character position, such as a diacritical mark and a basic letter.

context
A set of character strings associated with each process, that ordinarily determines which element (if any) from any given *context-dependent file* is accessed by that process. The *context* contains elements such as *cnode*name, the hardware attributes of the *cnode*(see *cluster node*), and possibly others. See *context*(5) for a complete list of items that can appear in the context.

This term applies to implementations that include the Series 300 Diskless Package.

context-dependent file (CDF)
A set of files, all associated with the same path name. The *context* of a process ordinarily determines which element (if any) from any given CDF will be accessed by that process. (See *cdf*(4)).

This term applies to implementations that include the Series 300 Diskless Package.

control character A character other than a graphic character that affects the recording, processing, transmission, or interpretation of text. In the *ASCII* character set, *control characters* are those in the range 0 through 31, and 127. Control characters can be generated by holding down [CTRL], [CONTROL], or [CNTL] (depending on what the control key is labeled on your terminal) and pressing a character key (as you would use SHIFT). These two-key sequences are often written as ctrl-d, for example, or ^D, where ^ stands for the control key. Both representations assume that the control key is held down while the second key is pressed.

controlling process
The session leader that establishes the connection to the *controlling terminal*. Should the terminal subsequently cease to be a controlling terminal for this session, the session leader ceases to be the controlling process.

controlling terminal
A terminal that is associated with a session. Each session can have at most one controlling terminal associated with it and a controlling terminal is associated with exactly one session. Certain input sequences from the controlling terminal cause signals to be sent to all processes in the foreground process group associated with the controlling terminal.

CS/80 or *CS-80* A family of mass storage devices that communicate with the controlling computer by means of a series of commands and data transfer protocol referred to as the *CS/80* (Command Set '80) command set. This command set was implemented in order to provide better forward/backward compatibility between models and generations of mass storage devices as technological advances develop. Some mass storage devices support only a subset of the full *CS/80* command set, and are usually referred to as *SS/80* (SubSet '80) devices. (See *cs80*(7).)

crash The unexpected shutdown of a program or system. If the operating system crashes, this is a "system crash", and requires the system to be re-booted.

current directory See *working directory*.

current working directory
See *working directory*.

daemon A process which runs in the background, and which is usually immune to termination instructions from a terminal. Its purpose is to perform various scheduling, clean-up, and maintenance jobs. *Lpsched*(1M) is an example of a *daemon*. It exists to perform these functions for line printer jobs queued by *lp*(1). An example of a permanent *daemon* (that is, one that should never die) is *cron*(1M).

data encryption A method for encoding information in order to protect sensitive or proprietary data. For example, all users' passwords are automatically encrypted by HP-UX. The encryption method used by HP-UX converts ASCII text into a base-64 representation using the alphabet ., /, 0-9, A-Z, a-z. See *passwd*(4) for the numerical equivalents associated with this alphabet.

default search path
The sequence of directory prefixes that *sh*(1), *time*(1), and other HP-UX commands apply in searching for a file known by an relative path name (that is, a path name not beginning with a *slash* (/)). It is defined by the environment variable **PATH** (see *environ*(5)). *Login*(1) sets **PATH** equal to **:/bin:/usr/bin**,

which means that your working directory is the first directory searched, followed by **/bin**, followed by **/usr/bin**. You can redefine the search path by modifying the value of **PATH**. This is usually done in **/etc/profile**, and/or in the **.profile** file found in your home directory.

delta A term used in the Source Code Control System (SCCS) to describe a unit of one or more textual changes to an SCCS file. Each time you edit an SCCS file, the changes you make to the file are stored separately as a *delta*. Then, using the *get*(1) command, you can specify which deltas are to be applied to or excluded from the SCCS file, thus yielding a particular version of the file. Contrast this with the *vi* or *ed* editor, which incorporates your changes into the file immediately, prohibiting you from obtaining a previous version of that file. See *SCCS, SCCS file*.

demon See *daemon*.

device A computer peripheral or an object that appears to an application as such.

device address See *bus address*.

device file See *special file*.

directory A file that provides the mapping between the names of files and their contents, and is manipulated by the operating system alone. For every file name contained in a directory, that directory contains a pointer to the file's *inode*; The pointer is called a *link*. A file can have several links appearing anywhere on the same file system. Each user is free to create as many directories as needed (using *mkdir*(1)), provided that the *parent directory* of the new directory gives the permission to do so. Once a directory has been created, it is ready to contain ordinary files and other directories. An HP-UX directory is named and behaves exactly like an ordinary file, with one exception: no user (including the super-user) is allowed to write data on the directory itself; this privilege is reserved for the HP-UX operating system.

By convention, a directory contains at least two links, . and .., referred to as *dot* and *dot-dot* respectively. *Dot* refers to the directory itself and *dot-dot* refers to its *parent directory*. A directory containing only . and .. is considered empty.

dot In any directory, the special file name, *dot*, consisting of a single dot character ("."), refers to the current directory. It can be used alone or at the beginning of a directory path name. (Also see *path name resolution*.) The *dot* also functions as a special command in the Bourne and Korn shells, and has special meaning in text editors and formatters, in parsing regular expressions and in designating file names.

dot-dot The special file name, *dot-dot* (".."), refers to the *parent directory*. If it begins a *path name*, *dot-dot* refers to the parent of the current directory. If it occurs in a path name, *dot-dot* refers to the parent directory of the directory preceding *dot-dot* in the path name string. As a special case, *dot-dot* refers to the current directory in any directory that has no parent (most often, the root directory). (Also see *path name resolution*.)

downshifting The conversion of an uppercase character to its lowercase representation.

effective group ID

Every process has an *effective group ID* that is used to determine *file access permissions*. A process's *effective group ID* is determined by the file (command) that process is executing. If that file's set-group-ID bit is set (located in the mode of the file, see *mode*), the process's *effective group ID* is set equal to the

file's group ID. This makes the process appear to belong to the file's group, perhaps enabling the process to access files which must be accessed in order for the program to execute successfully. If the file's set-group-ID bit is not set, the process's *effective group ID* is inherited from the process's parent. The setting of the process's *effective group ID* lasts only as long as the program is being executed, after which the process's effective group ID is set equal to its real group ID. See *group*, *real group ID*, and *set-group-ID bit*.

effective user ID A process has an *effective user ID* that is used to determine *file access permissions* (and other permissions with respect to system calls, if the effective user ID is 0, which means super-user). A process's effective user ID is determined by the file (command) that process is executing. If that file's set-user-ID bit is set (located in the mode of the file, see *mode*), the process's effective user ID is set equal to the file's user ID. This makes the process appear to be the file's owner, enabling the process to access files which must be accessed in order for the program to execute successfully. (Many HP-UX commands which are owned by *root*, such as *mkdir* and *mail*, have their set-user-ID bit set so other users can execute these commands.) If the file's set-user-ID bit is not set, the process's effective user ID is inherited from that process's parent. See *real user ID* and *set-user-ID bit*.

end-of-file (EOF) (1) The data returned when attempting to read past the logical end of a file via *stdio*(3S) routines. In this case end-of-file is not properly a character.

(2) The character [CTRL]-[D].

(3) A character defined by *stty*(1) or *ioctl*(2) (see *termio*(7)) to act as end-of-file on your terminal. Usually this is [CTRL]-[D].

(4) The return value from *read*(2) that indicates end of data.

environment The set of defined shell variables (some of which are PATH, TERM, SHELL, EXINIT, HOME) that define the conditions under which your commands run. These conditions can include your terminal characteristics, home directory, and default search path. Each shell variable setting in the current process is passed on to all *child processes* that are created, provided that each shell variable setting has been exported via the *export* command (see *sh*(1)). Unexported shell variable settings are meaningful only to the current process, and any child processes created get the default settings of certain shell variables by executing **/etc/profile**, **$HOME/.profile**, or **$HOME/.login**.

Epoch The Epoch refers to the time at 0 hours, 0 minutes, 0 seconds, Coordinated Universal Time on January 1, 1970. Increments quantify the amount of time elapsed from the Epoch to the referenced time.

Leap seconds, which occur at irregular intervals, are not reflected in the count of seconds between the Epoch and the referenced time. (Fourteen leap seconds occurred in the years 1970 through 1988.)

FIFO special file (FIFO)
A type of *file*. Data written to a *FIFO* is read on a first-in-first-out basis. Other characteristics are described under *open*(2), *read*(2), *write*(2) and *lseek*(2).

file A stream of bytes that can be written to and/or read from. A *file* has certain attributes, including permissions and type. File types include *regular file*, *character special file*, *block special file*, *FIFO special file*, *context-dependent file*, network special file, *directory*, and *symbolic link*. Every file must have a *file name* that enables the user (and many of the HP-UX commands) to refer to the contents of the file. The system imposes no particular structure on the contents of

a file, although some programs do. Files can be accessed serially or randomly (indexed by byte offset). The interpretation of file contents and structure is up to the programs that access the file.

file access mode A characteristic of an *open file description* that determines whether the described file is open for reading, writing, or both. (See *open*(2).)

file access permissions

Every file in the *file hierarchy* has a set of access permissions. These permissions are used in determining whether a process can perform a requested operation on the file (such as opening a file for writing). Access permissions are established when a file is created via the *open*(2) or *creat*(2) system calls, and can be changed subsequently through the *chmod*(2) call. These permissions are read by *stat*(2) or *fstat*(2).

File access controls whether a file can be read, written, or executed. Directory files use the execute permission to control whether or not the directory can be searched.

File access permissions are interpreted by the system as they apply to three different classes of users: the *owner* of the file, the users in the file's *group*, and anyone else ("other"). Every file has an independent set of access permissions for each of these classes. When an access check is made, the system decides if permission should be granted by checking the access information applicable to the caller.

Read, write, and execute/search permissions on a file are granted to a process if any of the following conditions are met:

The process's *effective user ID* is super-user.

The process's *effective user ID* matches the user ID of the owner of the file and the appropriate access bit of the *owner* portion (0700) of the file mode is set.

The process's *effective user ID* does not match the user ID of the owner of the file, and either the process's *effective group ID* matches the group ID of the file, or the group ID of the file is in the process's group access list, and the appropriate access bit of the *group* portion (070) of the file mode is set.

The process's *effective user ID* does not match the user ID of the owner of the file, and the process's *effective group ID* does not match the group ID of the file, and the group ID of the file is not in the process's group access list, and the appropriate access bit of the "other" portion (07) of the file mode is set.

Otherwise, the corresponding permissions are denied.

file descriptor A small unique, per-process, non-negative integer identifier that is used to refer to a file opened for reading and/or writing. Each *file descriptor* refers to exactly one *open file description*.

A *file descriptor* is obtained through system calls such as *open*(2), *creat*(2), *dup*(2), *fcntl*(2) or *pipe*(2). The *file descriptor* is used as an argument by calls such as *read*(2), *write*(2), *ioctl*(2), and *close*(2).

The value of a *file descriptor* has a range from 0 to one less than the system-defined maximum. The system-defined maximum is the value NOFILE in **<sys/param.h>**.

file group class A process is in the *file group class* of a file if the process is not the *file owner class* and if the *effective group ID* or one of the *supplementary group IDs* of the process matches the group ID associated with the file.

file hierarchy The collection of one or more *file systems* available on a system. All *files* in these *file systems* are organized in a single hierarchical structure in which all of the non-terminal nodes are *directories*. Because multiple *links* can refer to the same *file*, the directory is properly described as a directed graph.

file name A string of up to 14 bytes (or 255 bytes on file systems configured to support long file names) used to refer to an ordinary file, special file, or directory. The byte values zero (null) and slash ("/") cannot be used as characters in a file name. Note that it is generally unwise to use *, ?, [, !, or] as part of file names because the shell attaches special meaning to these characters (see *sh*(1), *csh*(1), or *ksh*(1)). Avoid beginning a file name with −, +, or =, because to some programs, these characters signify that a command argument follows. The last byte of a file name should not be a "+" due to interactions with context-dependent files on HP Diskless systems. A file name is sometimes called a path name component. Although permitted, it is inadvisable to use characters that do not have a printable graphic on the hardware you commonly use, or that are likely to confuse your terminal.

file name portability
File names should be constructed from the *portable file name character set* because the use of other characters can be confusing or ambiguous in certain contexts.

file offset The file offset specifies the position in the file where the next I/O operation begins. Each *open file description* associated with either a regular file or special file has a *file offset*. There is no file offset specified for a *pipe* or *FIFO*.

file other class A process is in the *file other class* if the process is not in the *file owner class* or *file group class.*

file owner class A process is in the *file owner class* if the *effective user ID* of the process matches the user ID of the file.

file permission bits
See *permission bits*.

file pointer A data element, obtained through any of the *fopen*(3S) standard I/O library routines that "points to" (refers to) a file opened for reading and/or writing, and which keeps track of where the next I/O operation will take place in the file (in the form of a byte offset relative to the beginning of the file). After obtaining the file pointer, it must thereafter be used to refer to the open file when using any of the standard I/O library routines. (See *stdio*(3S) for a list of these routines.)

file serial number
A *file serial number* is a file-system-unique identifier for a given file, and is also known as the file's *inode number*. Each *file serial number* identifies exactly one *inode*. *File serial numbers* are not necessarily unique across *file systems* in the *file hierarchy* .

file status flags A characteristic of an *open file description*. These flags can be used to modify the behavior of system calls that access the file described by the *open file description*.

file system A collection of *files* and supporting data structures residing on a mass storage volume. A file system provides a name space for *file serial numbers* referring to those files. Refer to the *System Administrator Manual* supplied with your system for details concerning file system implementation and maintenance.

file times update Each file has three associated time values that are updated when file data is accessed or modified, or when the file status is changed. These values are returned in the file characteristics structure, as described in **<sys/stat.h>**. For each function in HP-UX that reads or writes file data or changes the file status, the appropriate time-related files are noted as "marked-for-update." When an update point occurs, any marked fields are set to the current time and the update marks are cleared. One such update point occurs when the file is no longer open for any process. Updates are not performed for files on *read-only file systems* .

filter A command that reads data from the standard input, performs a transformation on the data, and writes it to the standard output.

foreground process group
Each session that has established a connection with a controlling terminal has exactly one process group of the session as a foreground process group of that controlling terminal. The foreground process group has certain privileges when accessing its controlling terminal that are denied to background process groups. See *read*(2) and *write*(2).

foreground process group ID
The process group ID of the foreground process group.

fork An HP-UX system call (*fork*(2)) which, when invoked by an existing process, causes a new process to be created. The new process is called the *child process;* the existing process is called the *parent process.* The child process is created by making an exact copy of the parent process. The parent and child processes are able to identify themselves by the value returned by their corresponding *fork* call (see *fork*(2) for details).

graphic character
A character other than a control character that has a visual representation when hand-written, printed, or displayed.

group An association of zero or more users who must all be permitted to access the same set of files. The members of a group are defined in the files **/etc/passwd** and **/etc/logingroup** (if it exists) via a numerical group ID that must be between zero and {UID_MAX}, inclusive. Users with identical group IDs are members of the same group. An ASCII group name is associated with each group ID in the file **/etc/group**. A group ID is also associated with every file in the *file hierarchy,* and the mode of each file contains a set of permission bits that apply only to this group. Thus, if you belong to a group that is associated with a file, and if the appropriate permissions are granted to your group in the file's mode, you can access the file. When the identity of a group is associated with a process, a group ID value is referred to as a real group ID, an effective group ID, one of the supplementary group IDs, or a saved set-group-ID. See *real group ID, effective group ID, privileged group , supplementary group ID* , and *set-group-ID bit.*

group access list The group access list is a set of *supplementary group IDs* used in determining resource accessibility. Access checks are performed as described in *file access permissions* .

hidden directory A specially marked *directory* used to implement *context-dependent files*. (see *cdf*(4)).

This term applies to implementations that include the Series 300 Diskless Package.

hierarchical directory
A directory (or file system) structure in which each directory may contain other directories as well as files.

home directory The directory name given by the value of the shell variable HOME. When you first log in, *login*(1) automatically sets HOME equal to your *login directory*. You may change its value at any time, however. This is usually done in the **.profile** file contained in your *login* directory. Setting HOME in no way affects your *login directory*, but simply gives you a convenient way of referring to what should be your most commonly used directory.

host name An *ASCII* string of at most 8 characters (of which only 6 are supported by all the various manufacturer's UNIX operating systems) which uniquely identifies an HP-UX system on a *uucp*(1) network. The *host name* for your system may be viewed and/or set with the *hostname*(1) command. Systems without a defined host name are described as "unknown" on the *uucp*(1) network. Do not confuse a host name with a *node name*, which is a string that uniquely identifies an HP-UX system on a Local Area Network (LAN). Although your host and node names may be identical, they are set and used by totally different software. See *node name*.

image The current state of your computer (or your portion of the computer, on a multi-user system) during the execution of a command. Often thought of as a "snapshot" of the state of the machine at any particular moment during execution.

init A *system process* that performs initialization, is the ancestor of every other process in the system, and is used to start *login* processes. *Init* usually has a *process ID* of 1.

interleave factor A number that determines the order in which sectors on a mass storage medium are accessed. It can be optimized to make data acquisition more efficient.

inode An *inode* is a structure that describes a file and is identified in the system by a *file serial number*. Every file or directory has associated with it an *inode*. Permissions that specify who can access the file and how are kept in a 9-bit field that is part of the *inode*. The *inode* also contains the file size, the user and group ID of the file, the number of links, and pointers to the disk blocks where the file's contents can be found. Each connection between an *inode* and its entry in one or more directories is called a *link*.

inode number See *file serial number*.

Internal Terminal Emulator (ITE)
The "device driver" code contained in the HP-UX kernel and associated with the computer's built-in keyboard and display or a particular keyboard and display connected to the computer, depending on the Series and Model of your HP-UX computer. See *system console* and the *System Adminstrator Manual* supplied with your system for details.

internationalization
The concept of providing software with the ability to support the *native*

language , *local customs* , and *coded character set* of the user.

interrupt signal The signal sent by SIGINT (see *signal*(2)). This signal generally terminates whatever program you are running. The key which sends this signal can be redefined with *ioctl*(2) or *stty*(1) (see *termio*(7)). It is often the ASCII DEL (rubout) character (the [DEL] key) or the [BREAK] key. [CONTROL]-[C] is often used instead.

intrinsic See *system call*.

I/O redirection A mechanism provided by the HP-UX shell for changing the source of data for standard input and/or the destination of data for standard output and standard error. See *sh*(1).

job control Job control allows users to selectively stop (suspend) the execution of processes and continue (resume) their execution at a later point.

The user employs this facility via the interactive interface jointly supplied by the system *tty* driver and either *csh*(1) or *ksh*(1). The *tty* driver recognizes a user-defined "suspend character" which causes the current foreground process group to stop and the user's job control shell to resume. The job control shell provides commands that continue stopped process groups in either the foreground or background. The *tty* also stops a background process group when any member of the background process group attempts to read from or write to the user's terminal. This allows the user to finish or suspend the *foreground process group* without interruption and continue the stopped *background process group* at a more convenient time.

See *csh*(1), *signal*(2), and *termio*(7).

kernel The HP-UX operating system. The kernel is the executable code responsible for managing the computer's resources, such as allocating memory, creating processes, and scheduling programs for execution. The kernel resides in RAM (Random Access Memory) whenever HP-UX is running.

LANG An environment variable that is used to inform a computer process of the user's requirements for *native language*, *local customs*, and *coded character set*.

library An *archive* file containing a set of subroutines and variables that can be accessed by user programs. For example, **/lib/libc.a** is a library containing all functions of section (2), and all functions of section (3) marked (3C) and (3S), in the HP-UX Reference. Similarly, **/lib/libm.a** is a library containing all functions in section (3) marked (3M) in the HP-UX Reference. See *intro*(3).

LIF An acronym for Logical Interchange Format. A standard format for mass storage implemented on many Hewlett-Packard computers to aid in media transportability. The *lif**(1) commands are used to perform various functions using LIF.

link *Link* is a synonym for directory entry. It is an object that associates a file name with any type of file. The information constituting a *link* includes the name of the file, and where the contents of that file may be found on a mass storage medium. One physical file may have several links to it. Several directory entries can associate names with a given file. If the links appear in different directories, the file may or may not have the same name in each. If the links appear in one directory, however, each link must have a unique name in that directory. Multiple links to directories are not allowed (except as created by the super-user). See *cp*(1), *link*(2), *unlink*(2), and *symbolic link*.

Also, to prepare a program for execution; see *linker*.

link count The *link count* of a file is the number of directory entries that refer to that file.

linker The *linker* combines one or more object programs into one program, searches libraries to resolve user program references, and builds an executable file in *a.out* format. This executable file is ready to be executed through the program loader, *exec*(2). The linker is invoked with the *ld*(1) command. The linker is often called a *link editor*.

local customs The conventions of a geographical area or territory for such things as date, time and currency formats.

localization

The process of adapting existing software to meet the local language, customs, and character set requirements of a particular geographical area.

logical block size The smallest unit of memory that can be allocated on a Series 500 SDF volume; a multiple of the physical sector size.

login The process of gaining access to HP-UX. This consists of successful execution of the *login* sequence defined by *login*(1) which varies depending on the system configuration. It includes providing a *login* name and possibly one or more passwords.

login directory The directory in which you are placed immediately after you log in. This directory is defined for each user in the file **/etc/passwd**. The shell variable HOME is set automatically to your *login directory* by *login*(1) immediately after you log in. See *home directory*.

magic number The first word of an *a.out*–format or archive file. This word contains the system ID, which states what machine (hardware) the file will run on, and the file type (executable, shareable executable, archive, etc.).

major number A number used exclusively to create special files that enable I/O to or from specific devices. This number indicates which device driver to use for the device. Refer to *mknod*(2) and the *System Administrator Manual* supplied with your system for details.

message catalog Program strings, such as program messages and prompts, are stored in a *message catalog* corresponding to a particular geographical area. Retrieval of a string from a *message catalog* is based on the value of the user's LANG environment variable (see *LANG*).

message queue identifier (msqid)

A *message queue identifier* is a unique positive integer created by a *msgget*(2) system call. Each *msqid* has a message queue and a data structure associated with it. The data structure is referred to as *msqid_ds* and contains the following members:

```
struct  ipc_perm msg_perm;/* operation permission */
ushort  msg_qnum;       /* number of msgs on q */
ushort  msg_qbytes;     /* max number of bytes on q */
ushort  msg_lspid;      /* pid of last msgsnd operation */
ushort  msg_lrpid;      /* pid of last msgrcv operation */
time_t  msg_stime;      /* last msgsnd time */
time_t  msg_rtime;      /* last msgrcv time */
time_t  msg_ctime;      /* last change time */
                        /* Times measured in secs since */
```

```
                                /* 00:00:00 GMT, Jan. 1, 1970 */
```

Message queue identifiers may be created using *stdipc*(3C).
Msg_perm is a ipc_perm structure that specifies the message operation permission (see below). This structure includes the following members:

```
ushort  cuid;       /* creator user id */
ushort  cgid;       /* creator group id */
ushort  uid;        /* user id */
ushort  gid;        /* group id */
ushort  mode;       /* r/w permission */
```

Msg_qnum is the number of messages currently on the queue. **Msg_qbytes** is the maximum number of bytes allowed on the queue. **Msg_lspid** is the process id of the last process that performed a *msgsnd* operation. **Msg_lrpid** is the process id of the last process that performed a *msgrcv* operation. **Msg_stime** is the time of the last *msgsnd* operation, **msg_rtime** is the time of the last *msgrcv* operation, and **msg_ctime** is the time of the last *msgctl*(2) operation that changed a member of the above structure.

message operation permissions

In the *msgop*(2) and *msgctl*(2) system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed interpreted as follows:

00400	Read by user
00200	Write by user
00060	Read, Write by group
00006	Read, Write by others

Read and Write permissions on a msqid are granted to a process if one or more of the following are true:

The process's effective user ID is super-user.

The process's effective user ID matches **msg_perm.[c]uid** in the data structure associated with *msqid* and the appropriate bit of the "user" portion (0600) of **msg_perm.mode** is set.

The process's effective user ID does not match **msg_perm.[c]uid** and either the process's effective group ID matches **msg_perm.[c]gid** or one of **msg_perm.[c]gid** is in the process's group access list and the appropriate bit of the "group" portion (060) of **msg_perm.mode** is set.

The process's effective user ID does not match **msg_perm.[c]uid** and the process's effective group ID does not match **msg_perm.[c]gid** and neither of **msg_perm.[c]gid** is in the process's group access list and the appropriate bit of the "other" portion (06) of **msg_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

metacharacter A character that has special meaning to the HP-UX shell, as well as to commands such as *ed*(1), *find*(1), and *egrep* (see *grep*(1)). The set of metacharacters includes: *, ?, !, [,], <, >, ;, |, ´, `, ", and &. Refer to *sh*(1) for the meaning associated with each. See also *regular expression.*

minor number A number that is an attribute of special files, specified during their creation and used whenever they are accessed, to enable I/O to or from specific devices. This number is passed to the device driver and is used to select which device in a family of devices is to be used, and possibly some operational modes. The exact format and meaning of the *minor number* is both system and driver

dependent. Refer to the *System Administrator Manual* supplied with your system for details.

On the Series 300 HP-IB devices, this number indicates the HP-IB *address*, *select code*, and the unit and/or volume numbers. (See *address*.) On the Series 800, a *minor number* is an index into a table in the *kernel*.

mode A 16-bit word associated with every file in the file system, stored in the *inode*. The least-significant 12 bits of the *mode* determine the read, write, and execute permissions for the file owner, file group, and all others, and contain the set-user-ID, set-group-ID, and "sticky" (save text image after execution) bits. The least-significant 12 bits can be set by the *chmod*(1) command if you are the file's owner or the super-user. The sticky bit can only be set by the super-user. These 12 bits are sometimes referred to as *permission bits*. The most-significant 4 bits specify the file type for the associated file and are set as the result of *open*(2) or *mknod*(2) system calls.

mountable file system
A removable blocked file system contained on some mass storage medium with its own root directory and an independent hierarchy of directories and files. See *block special file* and *mount*(1M).

multi-user state The condition of the HP-UX operating system in which terminals (in addition to the system console) allow communication between the system and its users. By convention, multi-user run level is set at state 2, which is usually defined to contain all the terminal processes and *daemons* needed in a multi-user environment. Run levels are table driven, and are specified by *init*(1M), which sets the run level by looking at the file **/etc/inittab.** Do not confuse the multi-user system with the multi-user state. A multi-user system is a system which may have more than one user actively communicating with the system when it is in the multi-user state. The multi-user state removes the single-user restriction imposed by the single-user state. (See *single-user state*, *inittab*(4).)

native language A computer user's spoken or written language, such as Dutch, English, French, German, Greek, Italian, Spanish, Swedish, Katakana, Turkish, Korean, Chinese, etc.

Native Language Support (NLS)
A feature of HP-UX that provides the user with internationalized software and the application programmer with tools to develop this software.

new-line The character with an ASCII value of 10 (line-feed) used to separate lines of characters. It is represented by \n in the C language and in various utilities. The terminal driver (see *tty*(7)) normally interprets the carriage-return/line-feed sequence sent by a terminal as a single new-line character.

NLSPATH
An environment variable used to indicate the search path for message catalogs (see *message catalog*).

node name A string of up to 31 characters, not including control characters or spaces, that uniquely identifies a node on a Local Area Network (LAN). The *node name* for each system is set by the *npowerup* command, which is one of the commands supplied with the optional LAN/9000 product. Do not confuse a node name with a *host name*, which is a string that uniquely identifies an HP-UX system on a *uucp* network. Your node and host names can be identical, but they are used and set by totally different software. See *host name*, *LAN/9000 User's Guide*, and *LAN/9000 Node Manager's Guide*.

non-spacing characters
Characters, such as a diacritical mark or accents, that are used in combination with other characters to form composite graphic symbols commonly found in non-English languages.

open file A file that is currently associated with a file descriptor.

open file description
An *open file description* records how a process or group of processes are accessing a file. Each *file descriptor* refers to exactly one *open file description*, but an *open file description* can be referred to by more than one file descriptor. The *file offset*, *file status flags*, and *file access modes* are attributes of an *open file description*.

ordinary file A type of HP-UX file containing ASCII text (e.g. program source), binary data (e.g. executable code), etc. Ordinary files can be created by the user through I/O redirection, editors, or HP-UX commands.

orphan process Whenever a *parent process* terminates for any reason and leaves behind one or more active *child* process, the child processes are called *orphan processes*. *Init*(1M) inherits (that is, becomes the effective parent of) all orphan processes.

orphaned process group
A process group in which the parent of every member is either itself a member of the group or is not a member of the group's session.

owner The owner of a file is usually the creator of that file. However, the ownership of a file can be changed by the super-user or the current owner with the *chown*(1) command or the *chown*(2) system call. The file owner is able to do whatever he wants with his files, including remove them, copy them, move them, change their contents, etc. He is also able to change the files' modes.

parent directory A directory's *parent directory* is the directory one level above it in the *file hierarchy*, with the exception of the directory entries for *dot* and *dot-dot*. All directories except the *root directory* (/) have one (and only one) parent directory.

parent process Whenever a new process is created by a currently-existing process (via *fork*(2)), the currently existing process is said to be the parent process of the newly created process. Every process has exactly one parent process (except the init process, see *init*), but each process can create several new processes with the *fork*(2) system call. The parent process ID of any process is the *process ID* of its creator.

parent process ID
A new process is created by a currently active process. The *parent process ID* of a process is the process ID of its creator for the lifetime of the creator. After the creator's lifetime has ended, the *parent process ID* is the process ID of *init*.

password A string of ASCII characters used to verify the identity of a user. Passwords can be associated with users and groups. If a user has a password, it is automatically encrypted and entered in the second field of that user's line in the **/etc/passwd** file. A user may create or change a password for himself with the *passwd*(1) command.

path name (Sometimes written as one word, *pathname*). A sequence of directory names separated by slashes, and ending with any file name. All file names except the last in the sequence *must* be directories. If a path name begins with a *slash* (/), it is an absolute path name (see *absolute path name*); otherwise it is a relative

path name (see *relative path name*). A path name defines the path to be followed through the hierarchical file system in order to find a particular file.

More precisely, a path name is a null-terminated character string constructed as follows:

<path-name>::=<file-name>|<path-prefix><file-name>|/
<path-prefix>::=<rtprefix>|/<rtprefix>
<rtprefix>::=<dirname>/|<rtprefix><dirname>/

where <file-name> is a string of one or more characters other than the ASCII slash and null, and <dirname> is a string of one or more characters (other than the ASCII slash and null) that names a directory. (File and directory names may consist of up to 14 characters on systems supporting short file names and up to 255 characters on systems supporting long file names.)

A *slash* (/) by itself names the *root directory*.

Unless specifically stated otherwise, the null or zero-length path name is treated as though it named a nonexistent file.

path name resolution

Path name resolution is performed for a process to resolve a path name to a particular file in a *file hierarchy*. Multiple path names can resolve to the same file, depending on whether resolution is sought in absolute or relative terms (see below). Each file name in the path name is located in the directory specified by its predecessor (for example, in the path name fragment "a/b", file "b" is located in directory "a"). *Path name resolution* fails if this cannot be accomplished.

If the path name begins with a slash, the predecessor of the first file name in the path name is understood to be the *root directory* of the process, and the path name is referred to as an *absolute path name.* If the path name does not begin with a slash, the predecessor of the first file name of the path name is understood to be the current working directory of the process, and the path name is referred to as a *relative path name.* A path name consisting of a single slash resolves to the root directory of the process.

path prefix A *path prefix* is a *path name* with an optional ending *slash* that refers to a *directory*.

permission bits The nine least-significant bits of a file's *mode* are referred to as file *permission bits*. These bits determine read, write, and execute permissions for the file's *owner*, the file's *group*, and all others. The bits are divided into three parts: owner, group and other. Each part is used with the corresponding file class of processes. The bits are contained in the file mode, as described in *stat*(5). The detailed usage of the file permission bits in access decisions is described in *file access permissions.*

pipe An interprocess I/O channel used to pass data between two processes. It is commonly used by the *shell* to transfer data from the standard output of one process to the standard input of another. On a command line, a pipe is signaled by a vertical bar (|). The output from the command(s) on the left of the vertical bar is channeled directly into the standard input of the command(s) on the right.

portable file name character set

The following set of graphical characters are portable across conforming implementations of IEEE Standard P1003.1:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
01234567890._-

The last three characters are the dot, underscore and hyphen characters, respectively. The hyphen should not be used as the first character of a portable file name.

privileged groups

A *privileged group* is a group that has had a *setprivgrp* (see *getprivgrp*(2)) operation performed on it, giving it access to some system calls otherwise reserved for the super–user.

proc1 See *init*.

process An invocation of a program, or the execution of an image (see *image*). Although all commands and utilities are executed within processes, not all commands or utilities have a one-to-one correspondence with processes. Some commands (such as *cd*) execute within a process, but do not create any new processes. Others (such as in the case of "ls | wc -l") create multiple processes. Several processes can be running in the same program, but each might have different data and be in different stages of execution. A process can also be thought of as an *address space* and single thread of control that executes within that address space and its required system resources. A *process* is created by another process issuing the *fork*(2) function. The process that issues *fork*(2) is known as the *parent process* and the new process created by the *fork*(2) as the *child process*.

process group Each process in the system is a member of a *process group*. This grouping permits the signaling of related processes. A newly created process joins the process group of its creator.

process group ID Each process group in the system is uniquely identified during its lifetime by a *process group ID*, a positive integer less than or equal to {PIC_MAX}. A *process group ID* cannot be reused by the system until the process group lifetime ends.

process group leader

A *process group leader* is a process whose process ID is the same as its process group ID.

process group lifetime

A period of time that begins when a *process group* is created and ends when the last remaining process in the group leaves the group, either due to process termination or by calling the *setsid*(2) or *setpgid*(2) functions.

process ID Each active process in the system is uniquely identified during its lifetime by a positive integer less than or equal to {PID_MAX} called a *process ID*. A process ID cannot be reused by the system until after the process lifetime ends. In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID cannot be reused by the system until the process group lifetime ends. A process that is not a system process shall not have a process ID of 1.

process lifetime After a process is created with a *fork*(2) function, it is considered active. Its thread of control and *address space* exist until it terminates. It then enters an inactive state where certain resources may be returned to the system, although

some resources, such as the *process ID* are still in use.
When another process executes a *wait*(2), *wait3*(2), or *waitpid*(2) function for an inactive process, the remaining resources are returned to the system. The last resource to be returned to the system is the process ID. At this time, the lifetime of the process ends.

program A sequence of instructions to the computer in the form of binary code (resulting from the compilation and assembly of program source).

prompt The character(s) displayed by the *shell* on the display indicating that the system is ready for a command. The prompt is usually a dollar sign ($) for ordinary users and a pound sign (#) for the super-user, but the user can redefine it to be any string by setting the shell variable **PS1** in his or her **.profile** file. See also *secondary prompt*.

quit signal The signal sent by SIGQUIT. See *signal*(2). The quit signal is generated by typing the character defined by the teletype handler as your quit signal. (See *stty*(1), *ioctl*(2), and *termio*(7).) The default is the ASCII FS character (ASCII value 28, generated by typing [CONTROL]-[\].) This signal usually causes a running program to terminate and generates a file containing the "core image" of the terminated process. The core image is useful for debugging purposes. (Some systems do not support core images, and on those systems no such file is generated.)

radix character The character that separates the integer part of a number from the fractional part. For example, in American usage, the *radix character* is a decimal point, while in Europe, a comma is used.

raw disk The name given to a disk for which there exists a *character special file* that allows direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O call.

read-only file system
A characteristic of a *file system* that prevents file system modifications.

real group ID A positive integer which is assigned to every user on the system. The association of a user and his or her *real group ID* is done in the file **/etc/passwd**. The modifier "real" is used because a user can also have an *effective group ID*. The real group ID can then be mapped to a group name in the file **/etc/group**, although it need not be. Thus, every user is a member of some group (which may be nameless), even if that group has only one member.

Every time a process creates a child process (via *fork*(2)), that process has a real group ID equal to the parent process's real group ID. This is useful for determining file access privileges within the process.

real user ID A positive integer which is assigned to every user on the system. A real user ID is assigned to every valid *login* name in the file **/etc/passwd**. The modifier "real" is used because a user can also have an *effective user ID* (see *effective user ID*).

Every time a process creates a child process (via *fork*(2)), that process has a real user ID equal to the parent process's real user ID. This is useful for determining file access privileges within the process.

regular expression
A string of zero or more characters that selects text. All the characters contained in the string might be literal, meaning that the regular expression

matches itself only; or one or more of the characters might be a *metacharacter*, meaning that a single regular expression could match several literal strings. Regular expressions are most often encountered in text editors (such as *ed*(1), *ex*(1), or *vi*(1)), where searches are performed for a specific piece of text, or in commands that were created to search for a particular string in a file (most notably *grep*(1)). Regular expressions are also encountered in the shell, *sh*(1), especially when referring to file names on command lines. See *ed*(1).

regular file A type of *file* that is a randomly accessible sequence of bytes, with no further structure imposed by the system. Its size can be extended. A regular file is also called an ordinary file.

relative path name
A *path name* that does not begin with a *slash* (/). It indicates that a file's location is given relative to your current *working directory*, and that the search begins there (instead of at the *root directory*). For example, **dir1/file2** searches for the directory **dir1** in your current working directory; then **dir1** is searched for the file **file2**.

root directory (1) The highest level directory of the hierarchical file system, from which all other files branch. In HP-UX, the *slash* (/) character refers to the *root directory*. The root directory is the only directory in the file system that is its own *parent directory*.

(2) Each process has associated with it a concept of a root directory for the purpose of resolving path name searches for those paths beginning with *slash* (/). A process's root directory need not be the root directory of the root file system, and can be changed by the *chroot*(1) command or *chroot*(2) system call. Such a directory appears to the process involved to have .. pointing to itself.

root server The node in a *cluster* to which the storage device containing the root file system of the *cluster* is physically attached.

This term applies to implementations that include the Series 300 Diskless Package.

root volume The mass storage volume which contains the boot area (which contains the HP-UX kernel) and the root directory of the HP-UX file system.

saved group ID Every process has a saved group ID that retains the process's *effective group ID* from the last successful *exec*(2) or *setresgid*(2), or from the last super-user call to *setgid*(2) or *setresuid*(2). *Setgid* permits a process to set its effective group ID to this remembered value. Consequently, a process that executes a program with the set-group-ID bit set and with a group ID of 5 (for example) can set its effective group ID to 5 at any time until the program terminates. See *exec*(2), *setuid*(2), *saved user ID*, *effective group ID*, and *set-group-ID bit*. The saved group ID is also known as the saved set-group-ID.

saved process group ID
Every process has a saved process group ID that retains the process's group ID from the last successful *exec*(2). See *setpgrp*(2), *termio*(7), and process group ID.

saved user ID Every process has a *saved user ID* that retains the process's *effective user ID* from the last successful *exec*(2) or *setresuid*(2), or from the last super-user call to *setuid*(2). *Setuid*(2) permits a process to set its effective user ID to this remembered value. Consequently, a process which executes a program with the set-user-ID bit set and with an owner ID of 5 (for example) can set its effective user ID to 5 at any time until the program terminates. See *exec*(2), *setuid*(2), *saved group ID*, *effective user ID*, and *set-user-ID bit*. The saved user

ID is also known as the saved set-user-ID.

SCCS An acronym for Source Code Control System. The Source Code Control System is a set of HP-UX commands which enable you to store changes to an SCCS file as separate "units" (called *deltas*). These units, each of which contains one or more textual changes to the file, can then be applied to or excluded from the SCCS file to obtain different versions of the file. The commands that make up SCCS are *admin*(1), *cdc*(1), *delta*(1), *get*(1), *prs*(1), *rmdel*(1), *sact*(1), *sccsdiff*(1), *unget*(1), *val*(1), and *what*(1). See *delta, SCCS file.*

SCCS file An ordinary text file which has been modified so that the Source Code Control System (SCCS) may be used with it. This modification is done automatically by the *admin*(1) command. See *SCCS, delta.*

SDF An acronym for Structured Directory Format. SDF was implemented on the obsolete Series 500 computers only, and provided tree-structured access to files through the root directory of the volume.

secondary prompt
One or more characters that the shell prints on the display, indicating that more input is needed. This prompt is much less often encountered than the shell's primary prompt (see *prompt*). When it occurs, it is usually caused by an omitted right quote on a string (which confuses the shell), or when you enter a shell programming language control-flow construct (such as a **for** construct) from the command line. By default, the shell's secondary prompt is the greater-than sign (>), but you can re-define it by setting the shell variable **PS2** appropriately in your **.profile** file.

select code On the Series 300, part of an *address* used for devices. A number determined by a setting on the interface card to which a peripheral device is connected, or by the particular I/O slot in which the I/O card resides. Multiple peripherals connected to the same interface card share the same select code.

semaphore identifier (semid)
A *semaphore identifier* is a unique positive integer created by a *semget*(2) system call. Each *semid* has a set of semaphores and a data structure associated with it. The data structure is referred to as *semid_ds* and contains the following members:

```
struct  ipc_perm sem_perm;/* operation permission */
ushort  sem_nsems;  /* number of sems in set */
time_t  sem_otime;  /* last operation time */
time_t  sem_ctime;  /* last change time */
                    /* Times measured in secs since */
                    /* 00:00:00 GMT, Jan. 1, 1970 */
```

Semaphore identifiers may be created using *stdipc*(3C).

Sem_perm is a ipc_perm structure that specifies the semaphore operation permission (see below). This structure includes the following members:

```
ushort  cuid;       /* creator user id */
ushort  cgid;       /* creator group id */
ushort  uid;        /* user id */
ushort  gid;        /* group id */
ushort  mode;       /* r/a permission */
```

The value of **sem_nsems** is equal to the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as a *sem_num*. Sem_num values run sequentially from 0 to the value of sem_nsems minus 1. **Sem_otime** is the time of the last *semop*(2) operation, and **sem_ctime** is the time of the last *semctl*(2) operation that changed a member of the above structure.

A semaphore is a data structure that contains the following members:

```
ushort  semval;     /* semaphore value */
short   sempid;     /* pid of last operation  */
ushort  semncnt;    /* # awaiting semval > cval */
ushort  semzcnt;    /* # awaiting semval = 0 */
```

Semval is a non-negative integer. **Sempid** is equal to the process ID of the last process that performed a semaphore operation on this semaphore. **Semncnt** is a count of the number of processes that are currently suspended awaiting this semaphore's **semval** to become greater than its current value. **Semzcnt** is a count of the number of processes that are currently suspended awaiting this semaphore's **semval** to become zero.

semaphore operation permissions

In the *semop*(2) and *semctl*(2) system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed interpreted as follows:

00400	Read by user
00200	Alter by user
00060	Read, Alter by group
00006	Read, Alter by others

Read and Alter permissions on a semid are granted to a process if one or more of the following are true:

The process's effective user ID is super-user.

The process's effective user ID matches **sem_perm.[c]uid** in the data structure associated with *semid* and the appropriate bit of the "user" portion (0600) of **sem_perm.mode** is set.

The process's effective user ID does not match **sem_perm.[c]uid** and the appropriate bit of the "group" portion (060) of **sem_perm.mode** is set.

The process's effective user ID does not match **sem_perm.[c]uid** and the process's effective group ID does not match **sem_perm.[c]gid** and neither of **sem_perm.[c]gid** is in the process's group access list and the appropriate bit of the "other" portion (06) of **sem_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

session

Each process group is a member of a 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 creator. A process can alter its session membership (see *setsid*(2)). A session can have multiple process groups (see *setpgid*(2)).

session leader A process that has created a session (see *setsid*(2)).

session lifetime The period between when a session is created and the end of the lifetime of all process groups that remain as members of the session.

set-group-ID bit A single bit in the mode of every file in the file system. If a file is executed whose set-group-ID bit is set, the effective group ID of the process which executed the file is set equal to the real group ID of the owner of the file. See *effective group ID, group,* and *real group ID.*

set-user-ID bit A single bit in the mode of every file in the file system. If a file is executed whose set-user-ID bit is set, the effective user ID of the process which executed the file is set equal to the real user ID of the owner of the file. See *effective user ID* and *real user ID.*

shared memory identifier

A shared memory identifier (shmid) is a unique positive integer created by a *shmget*(2) system call. Each shmid has a segment of memory (referred to as a shared memory segment) and a data structure associated with it. The data structure is referred to as *shmid_ds* and contains the following members:

```
struct  ipc_perm shm_perm;/* operation permission struct */
int     shm_segsz;        /* size of segment */
ushort  shm_cpid;         /* creator pid */
ushort  shm_lpid;         /* pid of last operation */
short   shm_nattch;       /* number of current attaches */
time_t  shm_atime;        /* last attach time */
time_t  shm_dtime;        /* last detach time */
time_t  shm_ctime;        /* last change time */
                          /* Times measured in secs since */
                          /* 00:00:00 GMT, Jan. 1, 1970 */
```

Shared memory identifiers may be created using *stdipc*(3C).

Shm_perm is a ipc_perm structure that specifies the shared memory operation permission (see below). This structure includes the following members:

```
ushort  cuid;     /* creator user id */
ushort  cgid;     /* creator group id */
ushort  uid;      /* user id */
ushort  gid;      /* group id */
ushort  mode;     /* r/w permission */
```

Shm_segsz specifies the size of the shared memory segment. **Shm_cpid** is the process id of the process that created the shared memory identifier. **Shm_lpid** is the process id of the last process that performed a *shmop*(2) operation. **Shm_nattch** is the number of processes that currently have this segment attached. **Shm_atime** is the time of the last *shmat* operation, **shm_dtime** is the time of the last *shmdt* operation, and **shm_ctime** is the time of the last *shmctl*(2) operation that changed one of the members of the above structure.

shared memory operation permissions

In the *shmop*(2) and *shmctl*(2) system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed interpreted as follows:

00400	Read by user
00200	Write by user
00060	Read, Write by group
00006	Read, Write by others

Read and Write permissions on a shmid are granted to a process if one or more of the following are true:

The process's effective user ID is super-user.

The process's effective user ID matches **shm_perm.[c]uid** in the data structure associated with *shmid* and the appropriate bit of the "user" portion (0600) of **shm_perm.mode** is set.

The process's effective user ID does not match **shm_perm.[c]uid** and either the process's effective group ID matches **shm_perm.[c]gid** or one of **shm_perm.[c]gid** is in the process's group access list and the appropriate bit of the "group" portion (060) of **shm_perm.mode** is set.

The process's effective user ID does not match **shm_perm.[c]uid** and the process's effective group ID does not match **shm_perm.[c]gid** and neither of **shm_perm.[c]gid** is in the process's group access list and the appropriate bit of the "other" portion (06) of **shm_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

shell The shell functions as both a command interpreter and an interpretive programming language. The shell is automatically invoked for every user who logs in, in order to provide a user-interface to the HP-UX operating system. See *sh*(1) *csh*(1), *ksh*(1), *pam*(1), and the tutorials supplied with your system for details.

shell program See *shell script.*

shell script A sequence of shell commands and shell programming language constructs stored in a file and invoked as a user command (program). No compilation is needed prior to execution, because the shell recognizes the commands and constructs that make up the shell programming language. A shell script is often called a *shell program* or a *command file.* See the shell programming article included in *HP-UX Selected Articles.*

signal Signals are software interrupts sent to processes, informing them of special situations or events. The term signal is also used to refer to the event itself. See *signal*(2).

single-user state A condition of the HP-UX operating system in which the system console provides the only communication mechanism between the system and its user. By convention, single-user state is usually specified by *init*(1M) as run-level **S** or **s**. Do not confuse the single-user state, in which the software is limiting a multi-user system to a single-user communication, with a single-user system, which can never communicate with more than one fixed terminal. (See *multi-user state.*)

slash The term *slash* is used to represent the literal character "/". This character is also known as "solidus". A *path name* consisting of a single slash resolves to the *root directory* of the process. (Also see *path name resolution.*)

source code The fundamental high-level information (program) written in the syntax of a specified computer language. Object (machine-language) code is derived from source code. When dealing with HP-UX shell command language, *source code*

is input to the command language interpreter. The term *shell script* is synonymous with this meaning. When dealing with the C Language, *source code* is input to the *cc*(1) command. *Source code* can also refer to a collection of sources meeting any of the above conditions.

special file Often called a *device file*, this is a file associated with an I/O device. Special files are read and written the same as ordinary files, but requests to read or write result in activation of the associated device. Due to convention and consistency, these files should always reside in the **/dev** directory.

special processes Processes with certain (small) process IDs are special. On a typical system, the ID's of 0, 1, and 2 are assigned as follows: Process 0 is the scheduler. Process 1 is the initialization process *init*, and is the ancestor of every other process in the system. It is used to control the process structure. On paging systems with virtual memory process 2 is the paging daemon.

standard error The destination of error and special messages from a program, intended to be used for diagnostic messages. The standard error output is often called *stderr*, and is automatically opened for writing on file descriptor 2 for every command invoked. By default, the user's terminal is the destination of all data written to *stderr*, but it can be redirected elsewhere. Unlike standard input and standard output, which are never used for data transfer in the "wrong" direction, standard error is occasionally read. This is not recommended practice, since I/O redirection is likely to break a program doing this.

standard input The source of input data for a program. The standard input file is often called *stdin*, and is automatically opened for reading on file descriptor 0 for every command invoked. By default, the user's terminal is the source of all data read from *stdin*, but it can be redirected from another source.

standard output The destination of output data from a program. The standard output file is often called *stdout*, and is automatically opened for writing on file descriptor 1 for every command invoked. By default, the user's terminal is the destination of all data written to *stdout*, but it can be redirected elsewhere.

stream A term most often used in conjunction with the standard I/O library routines documented in section (3) of this manual. A stream is simply a file pointer (declared as **FILE *stream**) returned by the *fopen*(3S) library routines. It may or may not have buffering associated with it (by default, buffering is assigned, but this may be modified with *setbuf*(3S)).

sticky bit A single bit in the mode of every file in the file system. If set, the contents of the file stay permanently in memory instead of being swapped back out to disk when the file has finished executing. Only the super-user can set the sticky bit. The sticky bit is read each time the file is executed (via *exec*(2)).

sub-directory A directory that is one (or perhaps more) levels lower in the file system hierarchy than a given directory. Sometimes called a *subordinate directory*.

subordinate directory
See *sub-directory*.

super block A block on each file system's mass storage medium which describes the file system. The contents of the super-block vary between implementations. Refer to the *System Administrator Manual* supplied with your system, and the appropriate *fs*(4) entry for details.

super-user The HP-UX system administrator. This user has access to all files, and can perform privileged operations. He has a real and effective user ID of 0, and, by

convention, the user name of *root*.

superior directory
See *parent directory*.

supplementary group ID
A process has up to {NGROUPS_MAX} supplementary group IDs used in determining file access permissions, in addition to the effective group ID. The supplementary group IDs of a process are set to the supplementary group IDs of the parent process when the process is created.

symbolic link A type of file that indirectly refers to a path name. See *symlink*(4).

system The term *system* is used to refer to the HP-UX operating system.

system asynchronous I/O
A method of performing I/O whereby a process informs a driver or subsystem that it wants to know when data has arrived or when it is possible to perform a write request. The driver or subsystem maintains a set of buffers through which the process performs I/O. See the *ioctl*(2), *select*(2), *read*(2), and *write*(2) manual pages for more information.

system call An HP-UX operating system kernel function available to the user through a high-level language (such as FORTRAN, Pascal, or C). Also called an "intrinsic" or a "system intrinsic." The available system calls are documented in section (2) of the *HP-UX Reference manual*.

system console A keyboard and display (or terminal) given a unique status by HP-UX and associated with the special file **/dev/console**. All boot ROM error messages, HP-UX system error messages, and certain system status messages are sent to the system console. Under certain conditions (such as the single-user state), the system console provides the only mechanism for communicating with HP-UX. See *HP-UX Selected Articles* and the *System Administrator Manual* provided with your system for details on configuration and use of the system console.

system process A *system process* is a process that runs on behalf of the system. It may have special implementation-defined characteristics.

terminal (or terminal device)
A *character special file* that obeys the specifications of *termio*(7).

tty Originally an abbreviation for teletypewriter, *tty* has come to mean *terminal*.

upshifting The conversion of a lowercase character to its uppercase representation.

user ID Each system user is identified by an integer known as a *user ID*, which is in the range of zero to {UID_MAX}, inclusive. Depending on how the user is identified with a process, a *user ID* value is referred to as a *real user ID*, an *effective user ID*, or a *saved set user ID*.

utility An executable file, which might contain executable object code (that is, a *program*), or a list of *commands* to execute in a given order (that is, a *shell script*). You can write your own utilities, either as executable programs or shell scripts (which are written in the shell programming language).

volume number Part of an address used for devices. A number whose meaning is software- and device-dependent, but which is often used to specify a particular volume on a multi-volume disk drive. See the *System Administrator Manual* supplied with your system for details.

white space One or more characters which when displayed cause a movement of the cursor or print head but do not result in the display of any visible graphic. The

white-space characters in the ASCII code set are space, tab, new-line, form-feed, carriage return and vertical tab. A particular command or routine might interpret some, but not necessarily all, white-space characters as delimiting fields, words, or command options.

working directory
Each process has associated with it the concept of a current working directory. For a shell, this appears as the directory in which you currently reside. This is the directory in which relative path name (i.e., a path name that does not begin with "/") searches begin. It is sometimes referred to as the *current directory*, or the *current working directory*.

zombie process The name given to a process which terminates for any reason, but whose parent process has not yet waited for it to terminate (via *wait*(2)). The process which terminated continues to occupy a slot in the process table until its parent process waits for it. Because it has terminated, however, there is no other space allocated to it either in user or kernel space. It is therefore a relatively harmless occurrence which will rectify itself the next time its parent process waits. The *ps*(1) command lists zombie processes as "defunct."

Index to Volume 3

Index
Volume 3

Description **Entry Name(Section)**

Description **Entry Name(Section)**

Description **Entry Name(Section)**

Description **Entry Name(Section)**