68NW9209H43A

SYSTEM V/88 Release 3.2 System Administrator's Reference Manual

MOTOROLA

MOTOROL

Motorola welcomes your

Manual Title ____________

Part Number ____________

Your Name ____________

Your Title ____________

Company ____________

Address ____________

____

____

General Information:

Do you read this manu

☐ Install the product

☐ Reference infor

In general, how do yo

☐ Index ☐ Table

Completeness: ☐ Ex

What topic would you

____

____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 2565 PHOENIX, ARIZONA

POSTAGE WILL BE PAID BY ADDRESSEE

MOTOROLA INC.
Computer Group, Microcomputer Division, DW164
2900 South Diablo Way
Tempe, AZ 85282-9741

Presentation: ☐ Excellent ☐ Very Good ☐ Good ☐ Fair ☐ Poor

What features of the manual are most useful (tables, figures, appendixes, index, etc.)?

Is the information easy to understand? ☐ Yes ☐ No If you checked no, please explain:

Is the information easy to find? ☐ Yes ☐ No If you checked no, please explain:

Technical Accuracy: ☐ Excellent ☐ Very Good ☐ Good ☐ Fair ☐ Poor

If you have found technical or typographical errors, please list them here.

Page Number	Description of Error

SYSTEM V/88 Release 3.2

System Administrator's

Reference Manual

(68NW9209H43A)

PREFACE

The *SYSTEM V/88 Release 3.2 System Administrator's Reference Manual* is for programmers and technical personnel who have a general familiarity with the SYSTEM V/88 operating system.

First Edition February 1990

CONTENTS

1M. Commands

NAME

intro – introduction to maintenance commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes. The commands in this section should be used along with those listed in Section 1 of the *User's Reference Manual* and Sections 1, 2, 3, 4, and 5 of the *Programmer's Reference Manual*. References of the form *name*(1), (2), (3), (4) and (5) refer to entries in the above manuals. References of the form *name*(1M), *name*(7) or *name*(8) refer to entries in this manual.

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.

SEE ALSO

getopt(1) in the *User's Reference Manual*.
getopt(3C) in the *Programmer's Reference Manual*.

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.

BUGS

Regrettably, not all commands adhere to the aforementioned syntax.

NAME

accept, reject – allow or 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 line printer (LP) 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* from accepting requests. This *reason* applies to all printers mentioned up to the next **–r** option. *Reason* is reported by *lp* when users direct requests to the named *destinations* and by *lpstat*(1). If the **–r** option is not present or the **–r** option is given without a *reason*, then a default *reason* will be used.

FILES

/usr/spool/lp/*

SEE ALSO

lpadmin(1M), lpsched(1M)
enable(1), lp(1), lpstat(1) in the *User's Reference Manual*.

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 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*(1).

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 from standard input that contain user ID, login name, and number of disk blocks and converts them to total accounting records, written to standard output, 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 filenames 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 alone turns process accounting off. 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*. 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
```

FILES

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

SEE ALSO

acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), diskusg(1M), fwtmp(1M), runacct(1M)
acctcom(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

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*(1). 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 filenames 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., SYSTEM V/68) style *acctcms* internal summary format records.

Since the output of **options -cjnst** is in internal-summary format, to see meaningful results, combine them with the **-a** option; this will give you ASCII output.

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), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M)
acctcom(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

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.

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. Hangup, 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 </etc/wtmp | sort +1n +2 > ctmp
acctcon2 <ctmp | acctmerg > ctacct
```

FILES

/etc/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), init(1M), runacct(1M)
acctcom(1), login(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

BUGS

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

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), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M)
acctcom(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

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), acctcon(1M), acctmerg(1M), acctsh(1M), cron(1M), fwtmp(1M), runacct(1M)
acctcom(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

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.

NAME

chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, 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"]

/usr/lib/acct/runacct [mmdd] [mmdd state]

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

/usr/lib/acct/startup

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

DESCRIPTION

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

ckpacct should be initiated via *cron*(1M). It periodically checks the size of **/usr/adm/pacct**. If the size exceeds *blocks* (1000 by default), *turnacct* is invoked with the 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/fstab.** If the **–o** flag is used, it will do a slower version of disk accounting by login directory. *files* specifies the one or more file system names where disk accounting will be done. If *files* are used, disk accounting will be done on these file systems only. If the **–o** flag is used, *files* should be mount points of mounted file systems. If omitted, they should be the special file names of mountable file systems.

lastlogin is invoked by *runacct* 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 be executed via *cron*(1M) on the first day of each month. *monacct* creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum.**

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

prctmp prints the session record file (normally **/usr/adm/acct/nite/ctmp** created by *acctcon*(1M)).

prdaily is invoked by *runacct* 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 specified 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; it may be used on current day's accounting data only.

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

runacct performs the accumulation of connect, process, fee, and disk accounting on a daily basis. It also creates summaries of command usage. For more information, see *runacct*(1M).

shutacct is invoked during a system shutdown to turn process accounting off and append a *reason* record to **/etc/wtmp**. The *reason* is limited to 11 characters.

startup can be called to turn the accounting on when the system is brought to a multi-user state.

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 *cron* and used to keep **pacct** to a reasonable size. *acct* starts and stops process accounting via *init* and *shutdown* accordingly.

FILES

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

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), cron(1M), diskusg(1M), fwtmp(1M), runacct(1M)
acctcom(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

NAME

adv – advertise a directory for remote access

SYNOPSIS

adv [-r] [-d *description*] *resource pathname* [*clients*...]
adv -m *resource* -d *description* | [*clients*...]
adv -m *resource* [-d *description*] | *clients*...
adv

DESCRIPTION

adv is the Remote File Sharing command used to make a resource from one computer available for use on other computers. The machine that advertises the resource is called the *server*, while computers that mount and use the resource are *clients*. (See *mount*(1M).) (A resource represents a directory, which could contain files, subdirectories, named pipes and devices.)

There are three ways **adv** is used: 1) to advertise the directory *pathname* under the name *resource* so it is available to Remote File Sharing *clients*; 2) to modify *client* and *description* fields for currently advertised resources; or 3) to print a list of all locally-advertised resources.

The following options are available:

–r
: Restricts access to the resource to a read-only basis. The default is read-write access.

–d *description*
: Provides brief textual information about the advertised resource. *description* is a single argument surrounded by double quotes (") and has a maximum length of 32 characters.

resource
: This is the symbolic name used by the server and all authorized clients to identify the resource. It is limited to a maximum of 14 characters and must be different from every other resource name in the domain. All characters must be printable ASCII characters but must not include periods (.), slashes (/), or white space.

pathname
: This is the local pathname of the advertised resource. It is limited to a maximum of 64 characters. This pathname cannot be the mount point of a remote resource and it can only be advertised under one resource name.

clients

These are the names of all clients that are authorized to remotely mount the resource. The default is that all machines that can connect to the server are authorized to access the resource. Valid input is of the form *nodename*, *domain.nodename*, *domain.*, or an alias that represents a list of client names. A domain name must be followed by a period (.) to distinguish it from a host name. The aliases are defined in **/etc/host.alias** and must conform to the alias capability in *mailx*(1).

-m *resource*

This option modifies information for a resource that has already been advertised. The resource is identified by a *resource* name. Only the *clients* and *description* fields can be modified. (To change the *pathname*, *resource* name, or read/write permissions, you must unadvertise and re-advertise the resource,)

When used with no options, *adv* displays all local resources that have been advertised; this includes the resource name, the pathname, the description, the read-write status, and the list of authorized clients. The resource field has a fixed length of 14 characters; all others are of variable length. Fields are separated by two white spaces, double quotes (") surround the description, and blank lines separate each resource entry.

This command may be used without options by any user; otherwise it is restricted to the super-user.

Remote File Sharing must be running before **adv** can be used to advertise or modify a resource entry.

EXIT STATUS

If there is at least one syntactically valid entry in the *clients* field, a warning will be issued for each invalid entry and the command will return a successful exit status. A non-zero exit status will be returned if the command fails.

ERRORS

If (1) the network is not up and running, (2) *pathname* is not a directory, (3) *pathname* isn't on a file system mounted locally, or (4) there is at least one entry in the *clients* field but none are syntactically valid, an error message will be sent to standard error.

FILES

/etc/host.alias

SEE ALSO

mount(1M), rfstart(1M), unadv(1M)
mailx(1) in the *User's Reference Manual*.

NAME

br – back up and restore files and/or file systems

SYNOPSIS

br

DESCRIPTION

The *br* command is a method of entering the backup and restore facility. It changes directory to **/backups** and runs the package as the user **br**, which has **root** permissions. The menus for the backup and restore facility are described in Procedures 5.4 and 5.5 and Chapter 5 of the *System Administrator's Guide.*

The identical function is available from the *sysadm* menu, i.e.:

sysadm backup_rest

The *br* command may (and should) be assigned a password. See *sysadm*(1), the *admpasswd* command.

SEE ALSO

sysadm(1)
Chapter 5 in the *System Administrator's Guide.*

NAME

brc, bcheckrc – system initialization procedures

SYNOPSIS

/etc/brc

/etc/bcheckrc

DESCRIPTION

These shell procedures are executed via entries in **/etc/inittab** by *init*(1M) whenever the system is booted (or rebooted).

The *bcheckrc* procedure executes the following steps:

1. Sets and exports the time zone (TZ) variable.
2. Prompts for the date if any of the following is true:

   There is no time-of-day clock,

   the time-of-day clock is invalid,

   explicit confirmation of the date is requested. (The variable **confirm-date** is assigned a value of zero by default. If an explicit date confirmation is needed every time the system is booted, a non-zero value must be assigned to this variable.)
3. Checks the status of the root file system. If the status of the root file system is found to be bad, it is automatically repaired.
4. Clears the mounted file system table **/etc/mnttab** and makes an entry for the root file system in the mount table.

The *brc* procedure copies the old advertise table **/etc/advtab** (related to RFS) to **/etc/oadvtab** and then creates a new **/etc/advtab** file with the appropriate permissions.

After these two procedures have executed, *init* checks for the *initdefault* value in **/etc/inittab**. This tells *init* in which run level to place the system. Since *initdefault* is initially set to **2**, the system will be placed in the multi-user state via the **/etc/rc2** procedure.

Note that *bcheckrc* should always be executed before *brc*. Also, these shell procedures may be used for several run-level states.

SEE ALSO

fsck(1M), init(1M), rc2(1M), shutdown(1M)

NAME

captoinfo – convert a termcap description into a terminfo description

SYNOPSIS

captoinfo [**–v** ...] [**–V**] [**–1**] [**–w** *width*] *file* ...

DESCRIPTION

captoinfo looks in *file* for *termcap* descriptions. For each one found, an equivalent *terminfo*(4) description is written to standard output, along with any comments found. A description which is expressed as relative to another description (as specified in the *termcap tc=* field) will be reduced to the minimum superset before being output.

If no *file* is given, then 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, then the file **/etc/termcap** is read.

–v
print out tracing information on standard error as the program runs. Specifying additional **–v** options will cause more detailed information to be printed.

–V
print out the version of the program in use on standard error and exit.

–1
cause the fields to print out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.

–w
change the output to *width* characters.

FILES

/usr/lib/terminfo/?/* compiled terminal description database

SEE ALSO

infocmp(1M), tic(1M)
curses (3X), terminfo(4) in the *Programmer's Reference Manual.*
Chapter 10 in the *Programmer's Guide*.

CAVEATS

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*) will sometimes produce a string which, though technically correct, may not be optimal. In particular, the rarely used *termcap* operation **%n** will produce strings that are especially long. Most occurrences of these non-optimal strings will be flagged with a warning message and may need to be recoded by hand.

The short two-letter name at the beginning of the list of names in a *termcap* entry, a hold-over from an earlier version of UNIX System V, has been removed.

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 the software which 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 was specified in the **ko** capability which could not be handled.

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*(1) 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 is given if the environment variable **TERM** is not set or is null.

cannot open *file* for reading.
The specified file could not be opened.

NOTES

captoinfo should be used to convert *termcap* entries to *terminfo*(4) entries because the *termcap* database (from earlier versions of UNIX System V) may not be supplied in future releases.

NAME

checkfsys – check a file system on a diskette

SYNOPSIS

checkfsys [**–p** *pattern*]

DESCRIPTION

The *checkfsys* command allows the user to check for and optionally repair a damaged file system on a diskette.

The user is asked one of the following three functions:

check the file system
No repairs are attempted.

repair it interactively
The user is informed about each instance of damage and asked if it should be repaired.

repair it automatically
The program applies a standard repair to each instance of damage.

The identical function is available under the *sysadm* menu:

sysadm checkfsys

The command may be assigned a password. See *sysadm*(1), the **admpasswd** subcommand.

WARNING

While automatic and interactive checks are generally successful, they can occasionally lose a file or a file's name. Files with content but without names are put in the **/file system/lost+found** directory.

If losing data is of particular concern, "check" the file system first to discover if it appears to be damaged. If it is damaged, use one of the repair mechanisms or the file system debugging utility, **fsdb**.

SEE ALSO

fsck(1M), fsdb(1M), makefsys(1M), mountfsys(1M), sysadm(1)

NAME

chroot – change root directory for a command

SYNOPSIS

/etc/chroot *newroot command*

DESCRIPTION

chroot causes the given command to be executed relative to the new root. The meaning of any initial slashes (/) in the path names is changed for the command and any of its child processes to *newroot*. Furthermore, upon execution, the initial working directory is *newroot*.

Notice, however, that if you redirect the output of the command to a file:

```
chroot newroot command >x
```

will create the file **x** relative to the original root of the command, not the new one.

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.

This command can be run only by the superuser.

SEE ALSO

cd(1) in the *User's Reference Manual*.
chroot(2) in the *Programmer's Reference Manual*.

BUGS

One should exercise extreme caution when referencing device files in the new root file system.

NAME

chrtbl – generate character classification and conversion tables

SYNOPSIS

chrtbl [*file*]

DESCRIPTION

The *chrtbl* command creates a character classification table and an upper/lower-case conversion table. The tables are contained in a byte-sized array encoded such that a table lookup can be used to determine the character classification of a character or to convert a character (see *ctype*(3C)). The size of the array is 257*2 bytes: 257 bytes are required for the 8-bit code set character classification table and 257 bytes for the upper- to lower-case and lower- to upper-case conversion table.

chrtbl reads the user-defined character classification and conversion information from *file* and creates two output files in the current directory. One output file, **ctype.c** (a C-language source file), contains the 257*2-byte array generated from processing the information from *file*. You should review the content of **ctype.c** to verify that the array is set up as you had planned. (In addition, an application program could use **ctype.c.)**

The first 257 bytes of the array in **ctype.c** are used for character classification. The characters used for initializing these bytes of the array represent character classifications that are defined in **/usr/include/ctype.h**; for example, **_L** means a character is lower case and **_S|_B** means the character is both a spacing character and a blank.

The last 257 bytes of the array are used for character conversion. These bytes of the array are initialized so that characters for which you do not provide conversion information will be converted to themselves. When you do provide conversion information, the first value of the pair is stored where the second one would be stored normally, and vice versa; for example, if you provide **<0x41 0x61>**, then **0x61** is stored where **0x41** would be stored normally, and **0x61** is stored where **0x41** would be stored normally.

The second output file (a data file) contains the same information, but is structured for efficient use by the character classification and conversion routines (see *ctype*(3C)). The name of this output file is the value of the character classification **chrclass** read in from *file*. This output file must be installed in the **/lib/chrclass** directory under this name by someone who is super-user or a member of group **bin**. This file must be readable by user, group, and other; no other permissions should be set. To use the character classification and conversion tables on this file, set the environmental variable **CHRCLASS** (see *environ*(5)) to the name of this file and export the variable; for example, if the name of this file (and character class) is **xyz**, you should issue the commands: **CHRCLASS=xyz ; export CHRCLASS .**

If no input file is given, or if the argument – is encountered, *chrtbl* reads from the standard input file.

The syntax of *file* allows the user to define the name of the data file created by *chrtbl*, the assignment of characters to character classifications and the relationship between upper- and lower-case letters. The character classifications recognized by *chrtbl* are:

chrclass
: name of the data file to be created by *chrtbl*.

isupper
: character codes to be classified as upper-case letters.

islower
: character codes to be classified as lower-case letters.

isdigit
: character codes to be classified as numeric.

isspace
: character codes to be classified as a spacing (delimiter) character.

ispunct
: character codes to be classified as a punctuation character.

iscntrl
: character codes to be classified as a control character.

isblank
: character code for the space character.

isxdigit
: character codes to be classified as hexadecimal digits.

ul
: relationship between upper- and lower-case characters.

Any lines with the number sign (#) in the first column are treated as comments and are ignored. Blank lines are also ignored.

A character can be represented as a hexadecimal or octal constant (for example, the letter **a** can be represented as 0x61 in hexadecimal or 0141 in octal). Hexadecimal and octal constants may be separated by one or more space and tab characters.

The dash character (-) may be used to indicate a range of consecutive numbers. Zero or more space characters may be used for separating the dash character from the numbers.

The backslash character (\) is used for line continuation. Only a carriage return is permitted after the backslash character.

The relationship between upper- and lower-case letters (**ul**) is expressed as ordered pairs of octal or hexadecimal constants: <*upper-case_character lower-case_character*>. These two constants may be separated by one or more space characters. Zero or more space characters may be used for separating the angle brackets (< >) from the numbers.

EXAMPLE

The following is an example of an input file used to create the ASCII code set definition table on a file named **ascii**.

```
chrclass  ascii
isupper   0x41 - 0x5a
islower   0x61 - 0x7a
isdigit   0x30 - 0x39
isspace   0x20 0x9 - 0xd
ispunct   0x21 - 0x2f    0x3a - 0x40  \
          0x5b - 0x60    0x7b - 0x7e
iscntrl   0x0 - 0x1f     0x7f
isblank   0x20
isxdigit  0x30 - 0x39    0x61 - 0x66  \
          0x41 - 0x46
ul       <0x41 0x61> <0x42 0x62> <0x43 0x63>  \
         <0x44 0x64> <0x45 0x65> <0x46 0x66>  \
         <0x47 0x67> <0x48 0x68> <0x49 0x69>  \
         <0x4a 0x6a> <0x4b 0x6b> <0x4c 0x6c>  \
         <0x4d 0x6d> <0x4e 0x6e> <0x4f 0x6f>  \
         <0x50 0x70> <0x51 0x71> <0x52 0x72>  \
         <0x53 0x73> <0x54 0x74> <0x55 0x75>  \
         <0x56 0x76> <0x57 0x77> <0x58 0x78>  \
         <0x59 0x79> <0x5a 0x7a>
```

FILES

/lib/chrclass/*
data file containing character classification and conversion tables created by *chrtbl*

/usr/include/ctype.h
header file containing information used by character classification and conversion routines

SEE ALSO

environron(5)
ctype(3C) in the *Programmer's Reference Manual* .

DIAGNOSTICS

The error messages produced by *chrtbl* are intended to be self-explanatory. They indicate errors in the command line or syntactic errors encountered within the input file.

NAME

ckbupscd – check file system backup schedule

SYNOPSIS

/etc/ckbupscd [**–m**]

DESCRIPTION

ckbupscd consults the file **/etc/bupsched** and prints the file system lists from lines with date and time specifications matching the current time. If the **–m** flag is present an introductory message in the output is suppressed so that only the file system lists are printed. Entries in the **/etc/bupsched** file are printed under the control of **cron**.

The System Administration commands *bupsched/schedcheck* are provided to review and edit the **/etc/bupsched** file.

The file **/etc/bupsched** should contain lines of 4 or more fields, separated by spaces or tabs. The first 3 fields (the schedule fields) specify a range of dates and times. The rest of the fields constitute a list of names of file systems to be printed if *ckbupscd* is run at some time within the range given by the schedule fields. The general format is:

time[,**time**] **day**[,**day**] **month**[,**month**] **fsyslist**

where:

time
Specifies an hour of the day (*0* through *23*), matching any time within that hour, or an exact time of day (*0:00* through *23:59*).

day
Specifies a day of the week (*sun* through *sat*) or day of the month (*1* through *31*).

month
Specifies the month in which the time and day fields are valid. Legal values are the month numbers (*1* through *12*).

fsyslist
The rest of the line is taken to be a file system list to print.

Multiple time, day, and month specifications may be separated by commas, in which case they are evaluated left to right.

An asterisk (*) always matches the current value for that field.

A line beginning with a sharp sign (#) is interpreted as a comment and ignored.

The longest line allowed (including continuations) is 1024 characters.

EXAMPLES

The following are examples of lines which could appear in the **/etc/bupsched** file.

06:00-09:00 fri 1,2,3,4,5,6,7,8,9,10,11 /applic
: Prints the file system name */applic* if *ckbupscd* is run between 6:00am and 9:00am any Friday during any month except December.

00:00-06:00,16:00-23:59 1,2,3,4,5,6,7 1,8 /
: Prints a reminder to backup the root (/) file system if *ckbupscd* is run between the times of 4:00pm and 6:00am during the first week of August or January.

FILES

/etc/bupsched specification file containing times and file system to back up

SEE ALSO

cron(1M)
echo(1), sh(1), sysadm(1) in the *User's Reference Manual*.

BUGS

ckbupscd will report file systems due for backup if invoked any time in the window. It does not know that backups may have just been taken.

NAME

clri – clear i-node

SYNOPSIS

/etc/clri *special i-number* ...

DESCRIPTION

clri writes nulls on the 64 bytes at offset *i-number* from the start of the i-node list. This effectively eliminates the i-node at that address. *Special* is the device name on which a file system has been defined. After *clri* is executed, any blocks in the affected file will show up as "not accounted for" when *fsck*(1M) is run against the file-system. The i-node may be allocated to a new file.

Read and write permission is required on the specified *special* device.

This command is used to remove a file which appears in no directory; that is, to get rid of a file which cannot be removed with the *rm* command.

SEE ALSO

fsck(1M), fsdb(1M), ncheck(1M)
fs(4) in the *Programmer's Reference Manual*.
rm(1) in the *User's Reference Manual*.

WARNINGS

If the file is open for writing, *clri* will not work. The file system containing the file should NOT be mounted.

If *clri* is used on the i-node number of a file that does appear in a directory, it is imperative to remove the entry in the directory at once, since the i-node may be allocated to a new file. The old directory entry, if not removed, continues to point to the same file. This sounds like a link, but does not work like one. Removing the old entry destroys the new file.

NAME

config – configure SYSTEM V/88

SYNOPSIS

/usr/src/uts/mot/cf/m88k/config [**–t**] [**–h** *file*] [**–c** *file*]
[**–m** *file*] *dfile*

DESCRIPTION

config is a program that takes a description of SYSTEM V/88 and generates two files. One file is a C program containing defines for the configurable system parameters (**config.h**). The second file is a C program defining the configuration tables for the various devices on the system and interrupt vector assignments (**conf.c**).

The **–h** option specifies the name of the configuration definition file; **config.h** is the default name.

The **–c** option specifies the name of the configuration table file; **conf.c** is the default name.

The **–m** option specifies the name of the file that contains all the information regarding supported devices; **/etc/master** is the default name. This file is supplied with SYSTEM V/88 and should not be modified unless you fully understands its construction.

The **–t** option requests a short table of major device numbers for character and block type devices. This can facilitate the creation of special files.

You must supply **dfile**; it must contain device information for your system. This file is divided into three parts. The first part contains physical device specifications. The second part contains system-dependent information. The third part contains microprocessor-specific information. The first two parts are required, the third part is optional. The format and contents of the **dfile** are provided in *dfile*(4) in the *Programmer's Reference Manual*. To obtain an example configuration, run the *sysdef*(1M) utility.

FILES

/etc/master	default input master device table
config.h	default output configuration definition file
conf.c	default output configuration table file

SEE ALSO

sysdef(1M)
dfile(4), master(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

Diagnostics are routed to the standard output and are self-explanatory.

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 administrator, the mode, owner, and group of the destination file will be that of the invoker.

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** is copied into **/bin** and given 0755, **bin, bin** as the mode, owner, and group, respectively.

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

cpset utilizes the file **/usr/src/destinations** to determine the final destination of a file. The locations file contains pairs of path names separated by spaces or tabs. The first name is the "official" destination (e.g., **/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** is:

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

When the actual installation happens, *cpset* verifies that the "old" path name 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 cases where cross generation is being done on a production system.

SEE ALSO

install(1M), mk(8)
make(1) in the *Programmer's Reference Manual*.

NAME

crash – examine system images

SYNOPSIS

/etc/crash [**–d** *dumpfile*] [**–n** *namelist*] [**–w** *outputfile*]

DESCRIPTION

The *crash* command is used to examine the system memory image of a live or a crashed system by formatting and printing control structures, tables, and other information. Command line arguments to *crash* are *dumpfile*, *namelist*, and *outputfile*.

Dumpfile is the file containing the system memory image. The default *dumpfile* is */dev/kmem*.

The text file *namelist* contains the symbol table information needed for symbolic access to the system memory image to be examined. The default *namelist* is **/unix**. If a system image from another machine is to be examined, the corresponding text file must be copied from that machine.

When the *crash* command is invoked, a session is initiated. The output from a *crash* session is directed to *outputfile*. The default *outputfile* is the standard output.

Input during a *crash* session is of the form:

function [*argument* ...]

where *function* is one of the *crash* functions described in the FUNCTIONS section of this manual page, and *arguments* are qualifying data that indicate which items of the system image are to be printed.

The default for process-related items is the current process for a running system and the process that was running at the time of the crash for a crashed system. If the contents of a table are being dumped, the default is all active table entries.

The following function options are available to *crash* functions wherever they are semantically valid.

–e
Display every entry in a table.

–f
Display the full structure.

–p
Interpret all address arguments in the command line as *physical* addresses.

–s *process*
Specify a process slot other than the default.

–w *file*
Redirect the output of a function to *file.*

Note that if the **–p** option is used, all address and symbol arguments explicitly entered on the command line will be interpreted as physical addresses. If they are not physical addresses, results will be inconsistent.

The functions *mode, defproc,* and *redirect* correspond to the function options **–p, –s,** and **–w.** The *mode* function may be used to set the address translation mode to physical or virtual for all subsequently entered functions; *defproc* sets the value of the process slot argument for subsequent functions; and *redirect* redirects all subsequent output.

Output from *crash* functions may be piped to another program in the following way:

```
function [argument ... ]!shell_command
```

For example,

mount ! grep rw

will write all mount table entries with an *rw* flag to the standard output. The redirection option (–w) cannot be used with this feature.

Depending on the context of the function, numeric arguments will be assumed to be in a specific radix. Counts are assumed to be decimal. Addresses are always hexadecimal. Table address arguments larger than the size of the function table will be interpreted as hexadecimal addresses; those smaller will be assumed to be decimal slots in the table. Default bases on all arguments may be overridden. The C conventions for designating the bases of numbers are recognized. A number that is usually interpreted as decimal will be interpreted as hexadecimal if it is preceded by *0x* and as octal if it is preceded by *0.* Decimal override is designated by *0d*, and binary by *0b.*

Aliases for functions may be any uniquely identifiable initial substring of the function name. Traditional aliases of one letter, such as *p* for *proc,* remain valid.

Many functions accept different forms of entry for the same argument. Requests for table information will accept a table entry number, a physical address, a virtual address, a symbol, a range, or an expression. A range of slot numbers may be specified in the form *a–b* where *a* and *b* are decimal numbers. An expression consists of two operands and an operator. An operand may be an address, a symbol, or a number; the operator may be +, –, *, /, &, or | . An operand which is a number should be preceded by a radix prefix if it is not a decimal number (*0* for octal, *0x* for hexidecimal, *0b* for binary). The expression must be enclosed in parentheses ( ). Other functions will accept any of these argument forms that are meaningful.

Two abbreviated arguments to *crash* functions are used throughout. Both accept data entered in several forms. They may be expanded into the following:

table_entry = table entry| address| symbol| range| expression

start_addr = address| symbol| expression

FUNCTIONS

? [**–w** file] List available functions.

!cmd
Escape to the shell to execute a command.

adv [**–e**] [**–w** file] [[**–p**] table_entry ...]
Print the advertise table.

base [**–w** file] number ...
Print *number* in binary, octal, decimal, and hexadecimal. A number in a radix other then decimal should be preceded by a prefix that indicates its radix as follows: *0x*, hexidecimal; *0*, octal; and *0b*, binary.

buffer [**–w** file] [–format] bufferslot

or

buffer [**–w** file] [–format] [**–p**] start_addr
Alias: **b**.
Print the contents of a buffer in the designated format. The following format designations are recognized: **–b**, byte: **–c**, character; **–d**, decimal; **–x**, hexadecimal; **–o**, octal; **–r**, directory; and **–i**, inode. If no format is given, the previous format is used. The default format at the beginning of a *crash* session is hexadecimal.

bufhdr [**–f**] [**–w** file] [[**–p**] table_entry ...]
Alias: **buf**.
Print system buffer headers.
The **–f** option produces different output depending on whether the buffer is local or remote (contains RFS data).

callout [**–w** file]
Alias: **c**.
Print the callout table.

dballoc [**–w** file] [class ...]
Print the dballoc table. If a class is entered, only data block allocation information for that class will be printed.

dbfree [**–w** file] [class ...]
Print free streams data block headers. If a class is entered, only data block headers for the class specified will be printed.

dblock [**–e**] [**–w** file] [**–c** class ...]

or

dblock [**–e**] [**–w** file] [[**–p**] table_entry ...]
Print allocated streams data block headers. If the class option (**–c**) is used, only data block headers for the class specified will be printed.

defproc [**–w** file] [**–c**]

or

defproc [**–w** file] [slot]
Set the value of the process slot argument. The process slot argument may be set to the current slot number (**–c**) or the slot number may be specified. If no argument is entered, the value of the previously set slot number is printed. At the start of a *crash* session, the process slot is set to the current process.

ds [**–w** file] virtual_address ...
Print the data symbol whose address is closest to, but not greater than, the address entered.

file [**–e**] [**–w** file] [[**–p**] table_entry ...]
Alias: **f**.
Print the file table.

findaddr [**–w** file] table slot
Print the address of *slot* in *table*. Only tables available to the *size* function are available to *findaddr*.

findslot [**–w** file] virtual_address ...
Print the table, entry slot number, and offset for the address entered. Only tables available to the *size* function are available to *findslot*.

fs [**–w** file] [[**–p**] table_entry ...]
Print the file system information table.

gdp [**–e**] [**–f**] [**–w** file] [[**–p**] table_entry ...]
Print the gift descriptor protocol table.

help [**–w** file] function ...
Print a description of the named function, including syntax and aliases.

inode [**–e**] [**–f**] [**–w** file] [[**–p**] table_entry ...]
Alias: **i**.
Print the inode table, including file system switch information.

lck [**–e**] [**–w** file] [[**–p**] table_entry ...]
Alias: **l**.
Print record locking information. If the **–e** option is used or table address arguments are given, the record lock list is printed. If no argument is entered, information on locks relative to inodes is printed.

linkblk [**–e**] [**–w** file] [[**–p**] table_entry ...]
Print the linkblk table.

major [**–w** file] [entry ...]
Print the MAJOR table.

map [**–w** file] mapname ...
Print the map structure of the given mapname.

mbfree [**–w** file]
Print free streams message block headers.

mblock [**–e**] [**–w** filename] [[**–p**] table_entry ...]
Print allocated streams message block headers.

mode [**–w** file] [mode]
Set address translation of arguments to virtual (v) or physical (p) mode. If no mode argument is given, the current mode is printed. At the start of a *crash* session, the mode is virtual.

mount [**–e**] [**–w** file] [[**–p**] table_entry ...]
  Alias: **m**.
  Print the mount table.

nm [**–w** file] symbol ...
  Print value and type for the given symbol.

od [**–p**] [**–w** file] [–format] [–mode] [–s process] start_addr [count]
  Alias: **rd**.
  Print *count* values starting at the start address in one of the following formats: character (**–c**), decimal (**–d**), hexadecimal (**–x**), octal (**–o**), ascii (**–a**), or hexadecimal/character (**–h**), and one of the following modes: long (**–l**), short (**–t**), or byte (**–b**). The default mode for character and ascii formats is byte; the default mode for decimal, hexadecimal, and octal formats is long. The format **–h** prints both hexadecimal and character representations of the addresses dumped; no mode needs to be specified. When format or mode is omitted, the previous value is used. At the start of a *crash* session, the format is hexadecimal and the mode is long. If no count is entered, 1 is assumed.

pfdat [**–e**] [**–w** file] [[**–p**] table_entry ...]
  Print the pfdata table.

proc [**–e**] [**–f**] [**–w** file] [[**–p**] table_entry ... #procid ...]

  or

proc [**–f**] [**–w** file] [**–r**]
  Alias: **p**.
  Print the process table. Process table information may be specified in two ways. First, any mixture of table entries and process ids may be entered. Each process id must be preceded by a **#**. Alternatively, process table information for runnable processes may be specified with the runnable option (**–r**).

qrun [**–w** file]
  Print the list of scheduled streams queues.

queue [**–e**] [**–w** file] [[**–p**] table_entry ...]
  Print streams queues.

quit
  Alias: **q**.
  Terminate the *crash* session.

rcvd [**–e**] [**–f**] [**–w** file] [[**–p**] table_entry ...]
Print the receive descriptor table.

redirect [**–w** file] [**–c**]

or

redirect [**–w** file] [file]
Used with a file name, redirects output of a *crash* session to the named file. If no argument is given, the file name to which output is being redirected is printed. Alternatively, the close option (**–c**) closes the previously set file and redirects output to the standard output.

region [**–e**] [**–f**] [**–w** file] [[**–p**] table_entry ...]
Print the region table.

search [**–p**] [**–w** file] [**–m** mask] [**–s** process] pattern start_addr length
Print the words in memory that match *pattern*, beginning at the start address for *length* words. The mask is anded (&) with each memory word and the result compared against the pattern. The mask defaults to 0xffffffff.

size [**–w** file] [**–x**] [structure_name ...]
Print the size of the designated structure. The (**–x**) option prints the size in hexadecimal. If no argument is given, a list of the structure names for which sizes are available is printed.

sndd [**–e**] [**–f**] [**–w** file] [[**–p**] table_entry ...]
Print the send descriptor table.

srmount [**–e**] [**–w** file] [[**–p**] table_entry ...]
Print the server mount table.

stack [**–w** file]
Print kernel stack backtrace

stat [**–w** file]
Print system statistics.

stream [**–e**] [**–f**] [**–w** file] [[**–p**] table_entry ...]
Print the streams table.

strstat [**–w** file]
Print streams statistics.

trace [**−w** file] [process]

or

trace [**−w** file] [[**−p**] **−i** start_addr]
Alias: **t**.
Print stack trace.

ts [**−w** file] virtual_address ...
Print closest text symbol to the designated address.

tty [**−e**] [**−f**] [**−w** file] [**−t** type [[**−p**] table_entry ...]]

or

tty [**−e**] [**−f**] [**−w** file] [[**−p**] start_addr]
Valid types: **pp**, **iu**.
Print the tty table. If no arguments are given, the tty table for both tty types is printed. If the **−t** option is used, the table for the single tty type specified is printed. If no argument follows the type option, all entries in the table are printed. A single tty entry may be specified from the start address.

user [**−f**] [**−w** file] [process]
Alias: **u**.
Print the ublock for the designated process.

var [**−w** file]
Alias: **v**.
Print the tunable system parameters.

vtop [**−w** file] [**−s** process] start_addr ...
Print the physical address translation of the virtual start address.

FILES

/dev/kmem system image of currently running system

BUGS

No facilities are provided for analyzing dumps of systems contained in other files unless that file is a copy of kernel memory. Even in that case, examining the user areas will not work properly in all cases.

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 in the directory **/usr/spool/cron/crontabs.** Users can submit their own *crontab* file via the *crontab*(1) command. Commands which are to be executed only once may be submitted via the *at*(1) command.

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

Since *cron* never exits, it should be executed only once. This is done routinely through **/etc/rc2.d/S75cron** at system boot time. /usr/lib/cron/FIFO is used as a lock file to prevent the execution of more than one *cron*.

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/FIFO	used as a lock file
/usr/lib/cron/log	accounting information
/usr/spool/cron	spool area

SEE ALSO

at(1), crontab(1), sh(1) in the *User's Reference Manual.*

DIAGNOSTICS

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

NAME

dcon - control dual console operation

SYNOPSIS

dcon [on] [off]

DESCRIPTION

The system utility *dcon* will initiate or terminate dual console operation via the *dcon on* and *dcon off* commands.

Initiating dual console operation via *dcon on*, causes I/O on the modem and console ports to become logically linked, i.e., all operations that are initiated on the console port also appear on the modem port and vice-versa.

Terminating dual console operation with *dcon off* will disconnect the communications link on the modem port, thus enabling the system administrator to maintain security.

Operators on either port can communicate with each other by executing *cat*(1) or *wall*(1).

WARNING

The dual console command can only be initiated by **root**.

NAME

dcopy – copy file systems for optimal access time

SYNOPSIS

/etc/dcopy [**–s**X] [**–an**] [**–d**] [**–v**] [**–ff***size*[:*isize*]] *inputfs outputfs*

DESCRIPTION

dcopy copies file system *inputfs* to *outputfs*. *Inputfs* is the device file for the existing file system; *outputfs* is the device file to hold the reorganized result. For the most effective optimization *inputfs* should be the raw device and *outputfs* should be the block device. Both *inputfs* and *outputfs* should be unmounted file systems (in the case of the root file system, the copy must be to a new pack).

With no options, *dcopy* copies files from *inputfs* compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap. The possible options are

–s*X*
supply device information for creating an optimal organization of blocks in a file. The forms of *X* are the same as the **–s** option of *fsck*(1M).

–a*n*
place the files not accessed in *n* days after the free blocks of the destination file system (default for *n* is 7). If no *n* is specified then no movement occurs.

–d
leave order of directory entries as is (default is to move sub-directories to the beginning of directories).

–v
currently reports how many files were processed, and how big the source and destination freelists are.

–f*fsize*[:*isize*]
specify the *outputfs* file system and inode list sizes (in blocks). If the option (or :*isize*) is not given, the values from the *inputfs* are used.

dcopy catches interrupts and quits, and reports on its progress. To terminate *dcopy* send a quit signal, followed by an interrupt or quit.

SEE ALSO

fsck(1M), mkfs(1M)
ps(1) in the *User's Reference Manual*.

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

option	*values*
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip (by reading) *n* input blocks before starting copy
iseek=*n*	seek *n* blocks from beginning of input file before copying
seek=*n*	seek *n* blocks from beginning of output file before copying
count=*n*	copy only *n* input blocks
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input block to *ibs*
. . . , . . .	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by **x** to indicate multiplication.

cbs is used only if *conv=ascii* or *conv=ebcdic* is specified. In the former case, *cbs* characters are placed into the conversion buffer (converted to ASCII). Trailing blanks are trimmed and a new-line added before sending the line to the output. In the latter case, ASCII characters are read into the conversion buffer (converted to EBCDIC). Blanks are added to make up an output block of size *cbs*.

After completion, *dd* reports the number of whole and partial input and output blocks.

DIAGNOSTICS

f+p blocks in(out)	numbers of full and partial blocks read(written)

NAME

ddefs – disk definition information manager

SYNOPSIS

/etc/ddefs [**-d** *ddefsdir*] **-n** [*diskname*]
/etc/ddefs [**-d** *ddefsdir*] **-erp** *diskname*
/etc/ddefs [**-d** *ddefsdir*] **-c** [*diskdefs*] [**-a** *altsdir*]

DESCRIPTION

The *ddefs* utility is used to add to or modify the information that describes disks. The disk definitions are contained in files in the **/etc/dskdefs** directory. The files in **/etc/dskdefs** are read by the *dinit*(1M) program to obtain format information about the disks.

To specify the directory, use the **–d** option followed by a directory name on the command line, as shown in the **SYNOPSIS** section. If no *ddefsdir* is given, the default is **/etc/dskdefs**.

The *diskname* is either a "device", specified as **/dev/rdsk/***devicename*, or a "*ddefs file*" in the directory *ddefsdir.*

The *ddefs* program provides several options for interactive use and non-interactive use:

-n [*diskname*]
(Interactive) Create a new disk definition where *diskname* corresponds to the *type* values listed for *dinit*(1M) (e.g., m323182 for a 182Mb CDC Winchester). If a name is given that already exists, the program automatically shifts to edit mode (**-e** option). If no *diskname* is given, the user is prompted for a name. When creating a new definition, *ddefs* will prompt for the name of a "template" disk. The template disk is usually one with similar, but not identical, attributes.

-e *diskname*
(Interactive) Edit an existing disk definition. If the *diskname* given does not exist, the program switches automatically to create mode (**-n** option).

-r *diskname*
(Interactive) Edit an existing disk definition (read-only).

-p *diskname*
(Non-interactive) Print an existing disk definition.

-c [*diskdefs*] [**-a** *altsdir*]
: (Non-interactive) Convert an old-style "diskdefs" file. If *diskdefs* is not given, the default is **/etc/diskdefs**. If *altsdir* is not given, the default is **/etc/diskalts**.

For every type of operation, the user has the option of specifying a directory for the disk definition file. If an existing definition is being edited, this is the directory where the definition file can be found. If a new definition is being created, this is the directory where the definition file will be placed. It is also the directory where the template can be found, if this feature is used.

Each disk definition is formatted as a series of lines, each line giving a parameter name, followed by a value. To modify a definition, move the cursor to the appropriate line and type the new value. When creating a new definition file, *ddefs* will not write the definition until all parameters are initialized (changed).

To obtain "help" information about a parameter, type **?** after the parameter name. The elements of the definition file and the help information for each are given in the following paragraphs. When *diskname* is a "*ddefs file*", all parameters are readable and writable. When *diskname* is a device, the following conventions are used to indicate how a parameter may be accessed: a single asterisk (*) following a parameter name indicates that the user has read access in interactive mode; a double asterisk (**) indicates that the parameter may be read and written in interactive mode; no asterisk indicates that the parameter is not applicable for a raw device (i.e., the parameter exists only in the *ddefs file*).

Comment
: This information is general comments. It cannot contain more than one line. Usually, the comment information is a description of the drive type supported by the definition file. If no comment is wanted, type **none**.

Disk type **
: This value is any unique integer that is used to identify the disk drive type. Each size drive on each controller should have a different type. Please refer to the manual page specific to each controller for information on each disk type value.

Format command
: The format program, if specified, is called by *dinit*(1M) to format a disk drive. The format program line should specify all options necessary to format the disk. If the driver is a new style driver (e.g., MVME327) then *dinit*(1M) can issue the format requests directly; enter **none**.

Diagnostic tracks *
: To reserve diagnostic tracks type **yes**. If not, type **no**.

Bad spot strategy *
: If the controller only supports perfect media, type **perfect**.

  BAD TRACKS: If bad track replacement is done by the SYSTEM V/88 bad track replacement software, type **software**. If the controller automatically performs bad track replacement, type **hardware**.

  BAD SECTORS: If the controller supports automatic bad sector replacement and requires the cylinder, head, and sector of each bad sector, then type **sector**.

  BAD SPOTS: If the controller supports automatic bad spot replacement and requires the cylinder, head, and byte offset of each bad spot, then type **spot**.

Maximum number of bad spots
: This value is the maximum number of bad spots expected on a disk of this type. This many alternate spots (sectors or tracks) will be allocated for this drive.

Number of sectors **
: This value is the total number of sectors on the disk drive.

Sector size (in bytes) **
: This value is the disk sector size (specified in bytes). Under current SYSTEM V/88 implementations, the sector size must be 128, 256, or 512 bytes.

Sectors per track **
: This value is the number of usable sectors per track on the formatted media.

Cylinders **
: This value is the total number of cylinders on the disk media.

Heads **
This value is the number of read/write heads on the drive. It is equivalent to the number of tracks per cylinder.

Precompensation cylinder **
This value is the disk cylinder number to start write precompensation.

Sector interleave **
This value is the sector interleave factor used during disk formatting. For no interleave (or one-to-one interleave), type **1**. Some controllers will automatically select an appropriate interleave factor when given an interleave value of **0**.

Spiral offset *
This value is the spiral offset applied when formatting disks. If no spiral offset is wanted, then type **0**.

Step rate **
This value is the seek step rate used when accessing the disk. Some controllers will automatically select an appropriate step rate when given a step rate of 0.

Starting head number **
This value is the starting head number of the drive. Most drives and controllers require a starting head number of zero.

ECC error length **
This value is the error correcting code data burst length.

Attributes mask (hex) **
This value is the disk attributes mask. Bits in this mask determine which bits in the attributes word are valid.

Extended attributes mask (hex) **
This value is the extended attributes mask.

Attributes word (hex) **

This value is the disk attributes word. The (hexadecimal) bit definitions are:

Name	Bit	Field Use	Bit Off	Bit On
ATWAS	0x0400	Alternate sectors?	no	yes
ATWFS	0x0200	Floppy size	5.25"	8"
ATWPC	0x0100	Precomp style	pre-write	post-read
ATWSK	0x0080	Seek after head change?	no	yes
ATWDD	0x0040	Track density of drive	single	double
ATWEN	0x0020	Encoding method	FM	MFM
ATWDT	0x0010	Disk type	floppy	hard
ATWSN	0x0008	Sector Numbering	Motorola	IBM
ATWDS	0x0004	Number of sides	single	double
ATWTD	0x0002	Track density of floppy	8" floppy	5.25" floppy
ATWMF	0x0001	Data density of medium	single	double

Extended attributes word (hex) **

This value is the extended attributes word.

Gap byte 1 (hex) **

This value is the first 'gap byte' required for formatting a disk. This parameter is controller-and drive-specific and may not be used by some controllers.

Gap byte 2 (hex) **

This value is the second 'gap byte' required for formatting a disk. This parameter is controller-and drive-specific and may not be used by some controllers.

Gap byte 3 (hex) **

This value is the third 'gap byte' required for formatting a disk. This parameter is controller-and drive-specific and may not be used by some controllers.

Gap byte 4 (hex) **

This value is the fourth 'gap byte' required for formatting a disk. This parameter is controller-and drive-specific and may not be used by some controllers.

Controller attribute (hex) **

This value is controller-and drive-specific information and may not be used by some controllers.

Unformatted sector size **
: This value is the unformatted sector size on the disk including the headers, gaps, ECC, and data. This parameter is controller and drive-specific, and may not be used by some controllers.

Sector slip count **
: This value is the sector slip count used while formatting to implement controller supported sector slipping. This count is the number of 'slip' sectors per track.

Slice count *
: This value is the dynamic slice count. If zero, then the driver will not use dynamic slicing. Legal non-zero slice counts are 8, 16, 32, 64, and 128. Some controllers only support a subset of these legal slice counts.

Root file system offset *
: This value is the 1024-byte block offset of the SYSTEM V/88 **root** file system.

Root file system size
: This value, if non-zero, is the size of the **root** file system (in 512-byte blocks) created on slice zero of the drive after it is formatted. If no file system is desired, type **0**.

/**usr** file system size
: This value, if non-zero, is the size of the /**usr** file system (in 512-byte blocks) created in slice one or two after the drive is formatted. If no /**usr** file system is desired, type **0**. The /**usr** file system is created only if a **root** file system is also created.

/**usr** file system slice
: This value specifies the slice for the /**usr** file system. This value must not be zero but may coincide with the swap slice.

Swap size
: This value, if non-zero, specifies the size of the system swap space (in 512-byte blocks). The swap space will be placed following the **root** file system if the swap slice is set to zero. If the swap slice and the /**usr** file system slice are both one, then swap will follow the /**usr** file system in slice one.

Swap slice
: This value specifies the swap slice. It may be set to zero (to share the **root** file system slice), or to one (to occupy its own slice or share with the /**usr** file system), or to two (to occupy its own slice following the /**usr** file system slice).

End-of-disk reserved area
: This value is the number of blocks reserved for system use at the end of the disk. This area may be used for bad track alternates or for diagnostic tracks.

Alternates list
: To add to the alternates list, type **add #[-#]**. To delete from the alternates list, type **delete #[-#]**. The '#' is any non-zero number. The sequence '#-#' specifies a range of non-zero numbers (including the end-points). To have no alternates, type **none**. The value printed in parentheses is the total number of alternates in the list.

When creating a disk description using *ddefs*, the following information must be given to have *dinit*(1M) create a slice table:

- A non-zero slice count; currently restricted to 0, 8, 16, 32, 64, 128.
- The **root** file system offset.
- The **root** file system size.
- The size of swap (if any).
- If swap is defined, what slice it resides on (allowable slices: 0, 1, 2).
- The /**usr** files system size (if any).
- If /**usr** is defined, what slice it resides on (allowable slices: 0, 1, 2).
- The size of the "end-of-disk" reserved area. This area is used for diagnostic and/or alternate tracks.

FILES

/etc/dskdefs/*	disk definition file

SEE ALSO

dinit(1M)

NAME

devnm – device name

SYNOPSIS

/etc/devnm [*names*]

DESCRIPTION

devnm identifies the special file associated with the mounted file system where the argument *name* resides.

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

EXAMPLE

The command:

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

produces

```
/dev/dsk/m323_0s2 /usr
```

if **/usr** is mounted on **/dev/dsk/m323_0s2**.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

brc(1M)

NAME

df – report number of free disk blocks and i-nodes

SYNOPSIS

df **[-ft|-b]** **[-l]** [*file-system* | *path-name* | *mounted-resource*]

DESCRIPTION

The **df** command prints out the number of free blocks and free i-nodes in file systems, path names, or mounted resources by examining the counts kept in the super-blocks.

file-system may be specified either by device name (e.g., **/dev/dsk/m323_0s0**) or by mount point directory name (e.g., **/usr**).

path-name can be a directory, regular file, or FIFO name. The report presents information for the device that contains the path name.

mounted-resource can be a remote resource name. The report presents information for the remote device that contains the resource.

If no arguments are used, the free space on all locally and remotely mounted file systems is printed.

The **df** command uses the following options:

–l
: reports on local file systems only.

–t
: causes the figures for total allocated blocks and i-nodes to be reported as well as the free blocks and i-nodes.

–f
: an actual count of the blocks in the free list is made, rather than taking the figure from the super-block (free i-nodes are not reported). This option will not print any information about mounted remote resources.

–b
: produces df output in the BSD 4.x format. File system information is reported in 512 byte blocks.

NOTE

If multiple remote resources are listed that reside on the same file system on a remote machine, each listing after the first one will be marked with an asterisk.

If the device on which a file system is mounted cannot be determined, the string "**UNKNOWN**" will be printed in place of the physical device name. This will occur whenever **df** tries to access a remote, networked file system or whenever **/etc/mnttab** is corrupted.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

mount(1M)
fs(4), mnttab(4) in the *Programmer's Reference Manual*.

NAME

dinit – disk initializer

SYNOPSIS

/etc/dinit [**–aefimnrRsTx**] [**–d** *desc*] [**–b** *file*] [**–t** *file*] *type rdev*

DESCRIPTION

dinit can be used to initialize specified disk *types*. The *type* must be a file in the directory **/etc/dskdefs**. Current values are shown in the following tables:

MVME323 Controller

DRIVE NAME	*type* Value
182Mb CDC Wren III ESDI (no sector slip)	m323182
182Mb CDC Wren III ESDI (sector slip)	m323182s
390Mb CDC Wren V ESDI (no sector slip)	m323390
390Mb CDC Wren V ESDI (sector slip)	m323390s

MVME327 Controller

DRIVE NAME	*type* VALUE
300Mb CDC 94171 Wren IV	m327cdcIV
600Mb CDC 94181 Wren V	m327cdcV
**Double-sided Double Density 5¼" Floppy	m327dsdd5
IBM PC/AT 5¼" 1.2Mb Floppy	m327pcat

** IBM 5¼" Format

For disk types with bad track handling, the alternate track numbers is taken from the file **/etc/dskdefs/***type*. There is no bad track support for floppy diskettes.

The *rdev* argument specifies a raw device, which must be of the form **/dev/r***string*. There must be a corresponding block device **/dev/***string* with the same minor device number as the character device. **DINIT MUST BE EXECUTED OVER THE SLICE REPRESENTING THE ENTIRE RAW DEVICE (i.e., SLICE 7).**

The following options are provided for *dinit*:

–r
Read bad spot list from disk and print it on stdout.

–a
Use with **–r** option to print alternates.

–x
Use with **–r** option to print in hexadecimal.

–n
Add a new bad spot. (See below).

–s
Skip saving and restoring of data when adding a new spot.

–f
Format disk. When formatting an unformatted disk, two read errors appear on the screen. These errors occur because the controller is trying to read configuration information from the disk. The messages can be ignored; the disk will be formatted as requested.

–m
Make file systems specified in **/etc/dskdefs/***type.* Slice table contains entries for slice 0 and slice (slice count -1); refer to *ddefs*(1M). This option only works when the **–f** option is used.

–i
To be used with the **–m** option. Inquire about gap size, number of inodes, blocks per cylinder. See *mkfs*(1M).

–R
Read manufacturer's defect list (only valid with **–f** option). NOTE: Currently, only the MVME323 supports this function.

–e
Use **EXORMACS** instead of **MOTOROLA** in sector 0, for compatibility with VM03 and EXORmacs bootloaders.

–d *desc*
Use *desc* as description string in sector 0.

–b *file*
Use *file* (a.out format) as the bootloader program.

–t *file*
Take bad track numbers from *file*, instead of interactively. This option only works when the **–f** option is used.

–T *file*
Take bad track numbers in track format; default is Head Cylinder.

Unless the **–f** option is given, *dinit* uses the parameters and bad spot information existing on the disk. Therefore, it is not necessary to re-enter bad track numbers on subsequent use of *dinit* on a disk. This is useful for changing the bootloader, description string. To change other disk parameters, refer to *ddefs*(1M).

The disk driver may occasionally report an I/O error for a bad spot not mapped out in the original format (see *errpt*(1M)). In such cases the new bad spot may be mapped without formatting the entire disk using the **–n** option only.

The information from *errpt*(1M), or the error message from the disk driver will contain the Head and Cylinder of the new bad spot. Enter this information when prompted by *dinit.*

dinit attempts to save the data from the new bad track unless the **–s** option is specified. If attempting to save data, expect I/O errors from the disk driver while *dinit* is executing.

FILES

/etc/dskdefs/*	disk definition file

SEE ALSO

ddefs(1M), errpt(1M).
System Administrator's Guide.

NAME

diskusg – generate disk accounting data by user ID

SYNOPSIS

diskusg [options] [files]

DESCRIPTION

diskusg generates intermediate disk accounting information from data in *files*, or the standard input if omitted. *diskusg* outputs lines to standard output, one per user, in the following format: *uid login #blocks*

where

uid
  is the numerical user ID of the user.

login
  is the login name of the user; and

#blocks
  is the total number of disk blocks allocated to this user.

diskusg normally reads only the i-nodes of file systems for disk accounting. In this case, *files* are the special filenames of these devices.

diskusg recognizes the following options:

–s
  specifies that the input data is already in *diskusg* output format, *not* special filenames of devices. *diskusg* combines all lines for a single user into a single line.

–v
  verbose. Prints a list on standard error of all files that are charged to no one.

–i *fnmlist*
  ignores 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 quotation marks. *diskusg* compares each name in this list with the file system name stored in the volume ID (see *labelit*(1M)).

–p *file*
  uses *file* as the name of the password file to generate login names. **/etc/passwd** is used by default.

–u *file*
writes records to *file* of those files that are charged to no one. Records consist of the special file name, the i-node 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)).

Note that although the intent of options **-v** and **-u** are the same, their outputs differ. For example,

diskusg -v /dev/usr writes `BAD UID: filesystem=/dev/usr, i-node=82, uid=105` to **stderr** whereas,

diskusg -u dtmp /dev/usr writes `/dev/usr 82 105` to file **dtmp**.

EXAMPLES

The following will generate daily disk accounting information for **root** on **/dev/dsk/m323_0s0**:

diskusg /dev/dsk/m323_0s0 | acctdisk > disktacct

In the next example, the "usr" file system is ignored despite its appearance later on the command line (i.e., **/dev/usr**):

diskusg -i "usr" /dev/usr /dev/dsk/m323_0s0

It is very important to enclose **"usr"** in quotation marks. Note that **"usr"** is derived from **"labelit /dev/usr"**, which displays the current file system name.

FILES

/etc/passwd	used for user ID to login name conversions

SEE ALSO

acct(1M), acctsh(1M)
acct(4) in the *Programmer's Reference Manual*.

NAME

dname – Print Remote File Sharing domain and network names

SYNOPSIS

dname [–D *domain*] [–N *netspec*] [**-dna**]

DESCRIPTION

dname prints or defines a host's Remote File Sharing (RFS) domain name or the network used by RFS as transport provider. When used with **d**, **n**, or **a** options, **dname** can be run by any user to print the domain name, network name or both, respectively. Only a user with root permission can use the **–D** *domain* option to set the domain name for the host or **–N** *netspec* to set the network specification used for RFS. (The value of *netspec* is the network device name, relative to the **/dev** directory. For example, the STARLAN NETWORK uses **starlan**.)

domain must consist of no more than 14 characters, consisting of any combination of letters (upper and lower case), digits, hyphens (–), and underscores (_)

When **dname** is used to change a domain name, the host's password is removed. The administrator will be prompted for a new password the next time RFS is started (*rfstart*(1M)).

If **dname** is used with no options, it will default to **dname –d**.

ERRORS

You cannot use the **–N** or **–D** options while RFS is running.

SEE ALSO

rfstart(1M)

NAME

du – summarize disk usage

SYNOPSIS

du [**–salrme**] [*names*]

DESCRIPTION

du reports the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, the current directory is used.

The optional arguments are:

–s
causes only the grand total (for each of the specified *names*) to be given.

–a
causes an output line to be generated for each file.

If neither **–s** or **–a** is specified, an output line is generated for each directory only.

–l
causes *du* to follow symbolic links and to report the size of the file to which the symbolic link points. (The default action is for *du* not to follow links. By default, *du* reports the size of the file containing the symbolic link.)

–r
will cause *du* to generate messages about directories that cannot be read, etc., rather than being silent (the default).

A file with two or more links is only counted once.

–m causes *du* to only count blocks for files located on the same mount device as the argument *names*.

–e causes *du* to return an exit code of 3 if *du* is unable to correctly deal with files that contain more than one link.

BUGS

If the **–a** option is not used, non-directories given as arguments are not listed.

If there are links between files in different directories where the directories are on separate branches of the file system hierarchy, *du* will count the excess files more than once.

Files with holes in them will get an incorrect block count. (See the chapter, *File System Administration*, in the *System Administrator's Guide*).

NAME

enpdl – ENP10-B (MVME330B) Download Program

SYNOPSIS

enpdl [**–rbsh**] [**–c** *cpu_number*]
enpdl [**–c** *cpu_number*] *download_file*

DESCRIPTION

The primary function of *enpdl* is to download *download_file* into an ENP10-B (MVME330B) board. This is done with a series of *ioctl*()s to reset the board, download the code, and start the code executing.

The following options are recognized by *enpdl*:

–c *cpu_number*
Specify cpu number to reset, download, or get board configuration from. The default cpu is 1 (one).

–r
Reset specified ENP10-B board. Combine the **–r** option with the **–c** option to reset a board other than the default cpu of 1.

–b
Show board configuration. This displays the following information for all ENP10-B boards in the system: cpu number, board type, board address, board number, and board state. If an ENP10-B board is in the system and does not show up in the display, it will not be recognized by the system. This usually happens because the board is strapped at the wrong address.

–h
When displaying board configuration, do not display the header.

–s
When displaying board configuration, show only the cpu number, board number, and board type.

For each entry in the **-b** display, the cpu number and board number are equivalent, and the board type is "330" (this information is used by other system facilities). The address is the VME base address of the board in hexadecimal. The board state is either "communicating" or "failure". A state of communicating means that the board is downloaded and responding to commands. Otherwise, the state is failure.

SEE ALSO

ifenp(7)

NAME

errdead – extract error records from dump

SYNOPSIS

/etc/errdead *dumpfile* [*namelist*]

DESCRIPTION

When hardware errors are detected by the system, an error record that contains information pertinent to the error is generated. If the error-logging daemon *errdemon*(1M) is not active or if the system crashes before the record can be placed in the error file, the error information is held by the system in a local buffer. *Errdead* examines a system dump, extracts such error records, and passes them to *errpt*(1M) for analysis.

The *dumpfile* specifies the file (or memory) that is to be examined. The system namelist is specified by *namelist*; if not given, **/unix** is used.

FILES

/unix	system namelist
/usr/bin/errpt	analysis program
/usr/tmp/errXXXXXX	temporary file

DIAGNOSTICS

Diagnostics may come from either *errdead* or *errpt*. In either case, they are intended to be self-explanatory.

SEE ALSO

errdemon(1M), errpt(1M)

NAME

errdemon – error-logging daemon

SYNOPSIS

/usr/lib/errdemon [*file*]

DESCRIPTION

The error logging daemon **errdemon** collects error records from the operating system by reading the special file **/dev/error** and places them in *file*. If *file* is not specified when the daemon is activated, **/usr/adm/errfile** is used. Note that *file* is created if it does not exist; otherwise, error records are appended to it, so that no previous error data is lost. No analysis of the error records is done by **errdemon**; that responsibility is left to *errpt*(1M). The error-logging daemon is terminated by using *errstop*(1M). Only the superuser may start the daemon, and only one daemon may be active at any time.

FILES

/dev/error	source of error records
/usr/adm/errfile	repository for error records

DIAGNOSTICS

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

SEE ALSO

errpt(1M), errstop(1M), err(7)
kill(1) in the *User's Reference Manual*.

NAME

errpt – process a report of logged errors

SYNOPSIS

errpt [**-a**] [**-d***dev*] [**-e***date*] [**-f**] [**-p***n*] [**-s***date*] [*files*]

DESCRIPTION

errpt processes data collected by the error logging mechanism (*errdemon*(1M)) and generates a report of that data. The default report is a summary of all errors posted in the files named. Options apply to all files and are described below. If no files are specified, *errpt* attempts to use **/usr/adm/errfile.**

A summary report notes the options that may limit its completeness, records the time stamped on the earliest and latest errors encountered, and gives the total number of errors of one or more types. Each device summary contains the total number of unrecovered errors, recovered errors, errors unabled to be logged, I/O operations on the device, and miscellaneous activities that occurred on the device. The number of times that *errpt* has difficulty reading input data is included as read errors.

Any detailed report contains, in addition to specific error information, all instances of the error logging process being started and stopped, and any time changes (via *date*(1)) that took place during the interval being processed. A summary of each error type included in the report is appended to a detailed report.

A report may be limited to certain records in the following ways:

–a
Produce a detailed report that includes all error types.

–dev
A detailed report is limited to data about devices given in *dev*, where *dev* can be one of two forms: a list of device identifiers separated from one another by a comma, or a list of device identifiers enclosed in double quotes and separated from one another by a comma and/or more spaces.

errpt is familiar with the common form of identifiers (see Section 7 of this volume). The devices for which errors are logged are system dependent. Additional identifiers are **int** and **mem** which include detailed reports of stray-interrupt and memory-parity type errors, respectively.

–e *date*
Ignore all records posted later than *date*, where *date* has the form *mmddhhmmyy*, consistent in meaning with the *date*(1) command.

–f
In a detailed report, limit the reporting of block device errors to unrecovered errors.

–p *n*
Limit the size of a detailed report to *n* pages.

–s *date*
Ignore all records posted earlier than *date*, where *date* has the form *mmddhhmmyy*, consistent in meaning with the *date*(1) command.

FILES

/etc/master	for configuration of devices in system
/usr/adm/errfile	default error file

SEE ALSO

date(1), errdead(1M), errdemon(1M), errfile(4)

BUGS

When illegal options are specified, *errpt* ignores them and generates default output.

NAME

fblkgen – convert track numbers into file system logical block numbers

SYNOPSIS

/etc/fblkgen [**-T**] [**-t** *file*] [**-d** *ddefsdir*] *devtype device*

DESCRIPTION

For each file system on the specified disk *device* which contains logical blocks residing on track numbers from the input, create a file which contains those logical block numbers, one per line.

By default, the input is assumed to be in head, cylinder format, where the head and cylinder are blank- or tab-separated, one pair per line. For example:

```
3 104
7 986
```

The files are placed in **/etc/badtracks**. The names are of the form **F.***x_ysp*, where *x* is the controller, *y* is the drive logical unit number on that controller, and *p* is the partition (or slice) in which this file system begins. No files are created for partitions which do not have bad blocks or which do not contain a file system.

These files are suitable as input files for *fsck*(-r option). If you do not remove existing files for the device before starting, the new numbers are appended.

fblkgen discards those input values having corresponding track numbers which are less than zero or larger than the maximum as computed from entries in **/etc/dskdefs/***devtype.*

If file systems appear to overlap because of an obsolete file system which has not yet had its superblock overwritten, you will be prompted to indicate which is the correct file system.

The options are:

–T
The input, whether from a file or entered interactively, is in track format, one track number per line. The default format without this option is head,cylinder pairs format.

–t *file*
Use the named file as input. The format is either single track numbers or head,cylinder pairs, one track number or pair per line. If this option is omitted, you will be prompted to enter the data manually.

–d ddefsdir
: Use the directory *ddefsdir* rather than **/etc/dskdefs** to find the entry for *devtype*.

devtype
: This is the device type found in **/etc/dskdefs** which uniquely identifies the device (e.g., m323182). See *ddefs*(1M).

device
: The disk device name of the form *x_y*, where *x* is the controller mnemonic and *y* is the drive logical unit number on that controller. For example, m323_1.

The *xformtrk*(1M) utility converts various input formats into a form suitable for **fblkgen**.

EXAMPLE

Use the list, which is in head,cylinder format, to generate a list of affected file system logical blocks on the first CDC WREN disk attached to the MVME323 controller.

```
fblkgen -t m323182 m323_0
```

FILES

/etc/badtracks/F.*device*
/etc/dskdefs/*

SEE ALSO

xformtrk(1M)

DIAGNOSTICS

Exit Codes:

0 - success	
1 - internal failure	unknown type or device, general error
2 - I/O failure	files not found, read/write failure, etc.
3 - bad command usage	syntax error within command line
4 - user interrupt	

NAME

ff – list file names and statistics for a file system

SYNOPSIS

/etc/ff **[-I]** **[-l]** **[-p***prefix***]** **[-s]** **[-u]** **[-a***n***]** **[-m***n***]** **[-c***n***]** **[-n***file***]** **[-i***i-node-list***]** *special*

DESCRIPTION

ff reads the i-list and directories of the *special* file, assuming it is a file system. I-node data is saved for files which match the selection criteria. Output consists of the path name for each saved i-node, plus other file information requested using the print *options* below. Output fields are positional. The output is produced in i-node order; fields are separated by tabs. The default line produced by *ff* is:

path-name i-number

With all *options* enabled, output fields would be:

path-name i-number size uid

The argument *n* in the *option* descriptions that follow is used as a decimal integer (optionally signed), where +*n* means more than *n*, –*n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hour period.

–I
Do not print the i-node number after each path name.

–l
Generate a supplementary list of all path names for multiply-linked files.

–p *prefix*
The specified *prefix* will be added to each generated path name. The default is **.** (dot).

–s
Print the file size, in bytes, after each path name.

–u
Print the owner's login name after each path name.

–a *n*
Select if the i-node has been accessed in *n* days.

–m *n*
Select if the i-node has been modified in *n* days.

–c *n*
Select if the i-node has been changed in *n* days.

–n *file*
Select if the i-node has been modified more recently than the argument *file*.

–i *i-node-list*
Generate names for only those i-nodes specified in *i-node-list*.

SEE ALSO

ncheck(1M)
find(1) in the *User's Reference Manual*.

BUGS

If the **–l** option is not specified, only a single path name out of all possible ones is generated for a multiply-linked i-node. If **–l** is specified, all possible names for every linked file on the file system are included in the output. However, no selection criteria apply to the names generated.

NAME

finc – fast incremental backup

SYNOPSIS

/etc/finc [**-a***n*] [**-m***n*] [**-c***n*] [**-n***file*] *file-system raw-tape*

DESCRIPTION

finc selectively copies the input *file-system* to the output *raw-tape* . The cautious will want to mount the input *file-system* read-only to insure an accurate backup, although acceptable results can be obtained in read-write mode. The tape must be previously labeled by *labelit*. The selection is controlled by the *options*, accepting only those inodes/files for whom the conditions are true.

It is recommended that production of a *finc* tape be preceded by the *ff* command, and the output of *ff* be saved as an index of the tape's contents. Files on a *finc* tape may be recovered with the *frec* command.

The argument *n* in the *selection-criteria* which follow is used as a decimal integer (optionally signed), where +*n* means more than *n*, –*n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hours.

–a *n*
True if the file has been accessed in *n* days.

–m *n*
True if the file has been modified in *n* days.

–c *n*
True if the i-node has been changed in *n* days.

–n *file*
True for any file which has been modified more recently than the argument *file*.

EXAMPLES

To write a tape consisting of all files from file-system /usr modified in the last 48 hours:

```
finc –m –2 /dev/rdsk/m323_0s2 /dev/rmt/m350_0
```

SEE ALSO

ff(1M), frec(1M), labelit(1M)
cpio(1) in the *User's Reference Manual*.

NAME

frec – recover files from a backup tape

SYNOPSIS

/etc/frec [**-p** *path*] [**-f** *reqfile*] *raw tape device i_number : name* . . .

DESCRIPTION

frec recovers files from the specified *raw tape device* backup tape written by *volcopy*(1M) or *finc*(1M), given their *i_numbers*. The data for each recovery request will be written into the file given by *name*.

The **–p** option allows you to specify a prefixing *path* different from your current working directory. This will be prefixed to any *names* that are not fully qualified, i.e., that do not begin with / or ./. If any directories are missing in the paths of recovery *names*, they will be created.

–p *path*
specifies a prefixing *path* to be used to fully qualify any names that do not start with / or ./.

–f *reqfile*
specifies a file which contains recovery requests. Each request should occupy one line in the file, with this format: *i_number:newname.*

EXAMPLES

To recover a file, i-number 1216 when backed-up, into a file named **junk** in your current working directory:

```
frec  /dev/rmt/ctape  1216:junk
```

To recover files with i_numbers 14156, 1232, and 3141 into files **/usr/src/cmd/a**, **/usr/src/cmd/b** and **/usr/joe/a.c**:

```
frec –p /usr/src/cmd /dev/rmt/ctape 14156:a 1232:b
  3141:/usr/joe/a.c
```

SEE ALSO

ff(1M), finc(1M), labelit(1M)
cpio(1) in the *User's Reference Manual*.

BUGS

While creating the intermediate directories contained in a pathname, *frec* can recover i-node fields only for those directories contained on the tape and requested for recovery.

NAME

fsba – file system block analyzer

SYNOPSIS

/etc/fsba [–b *target_block_size*] *file-system1* [*file-system2* ...]

DESCRIPTION

The *fsba* command determines the disk space required to store the data from an existing file system in a new file system with the specified logical block size. Each *file-system* listed on the command line refers to an existing file system and should be specified by device name (e.g., **/dev/rdsk/m323_0s2**).

The *target_block_size* specifies the logical block size in bytes of the new file system. Valid target block sizes are 1024, 2048, 4096, and 8192. Default target block size is 1024.

The *fsba* command prints information about how many 512-byte disk sectors are allocated to store the data in the old (existing) file system and how many would be required to store the same data in a new file system with the specified logical block size. It also prints the number of allocated and free i-nodes for the existing file system.

If the number of free sectors listed for the new file system is negative, the data will not fit in the new file system unless the new file system is larger than the existing file system. The new file system must be made at least as large as the number of sectors listed by *fsba* as allocated for the new file system. The maximum size of the new file system is limited by the size of the disk partition used for the new file system.

Note that it is possible to specify a *target_block_size* that is smaller than the logical block size of the existing file system. In this case the new file system would require fewer sectors to store the data.

SEE ALSO

mkfs(1M)

NAME

fsck, dfsck – check and repair file systems

SYNOPSIS

/etc/fsck [**–y**] [**–n**] [**–s**X] [**–S**X] [**–t***file*] [**–q**] [**–D**] [**–f**] [**–b**] [**–r***file*] [*file-systems*]
/etc/dfsck [*options1*] *fsys1* ... – [*options2*] *fsys2* ...

DESCRIPTION

fsck

fsck audits and interactively repairs inconsistent conditions for file systems. If the file system is found to be consistent, the number of files, blocks used, and blocks free are reported. If the file system is inconsistent the user is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data loss may be determined from the diagnostic output. The default action for each correction is to wait for the user to respond **yes** or **no**. If the user does not have write permission *fsck* defaults to a **–n** action.

The following options are accepted by *fsck*:

–y
Assume a **yes** response to all questions asked by *fsck*.

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

–s*X*
Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The
–s*X* option allows for creating an optimal free-list organization.

If *X* is not given, the values used when the file system was created are used. The format of *X* is *cylinder size:gap size*.

–SX
: Conditionally reconstruct the free list. This option is like –sX above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **–S** will force a **no** response to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.

–t
: If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **–t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **–t** flag, *fsck* will prompt the user for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

–q
: Quiet *fsck*. Do not print size-check messages. Unreferenced **fifos** will silently be removed. If *fsck* requires it, counts in the superblock will be automatically fixed and the free list salvaged.

–D
: Directories are checked for bad blocks. Useful after system crashes.

–f
: Fast check. Check block and sizes and check the free list. The free list will be reconstructed if it is necessary.

–b
: Reboot. If the file system being checked is the root file system and modifications have been made, then either remount the root file system or reboot the system. A remount is done only if there was minor damage.

–r
: The named *file* is assumed to contain file system block numbers that have been corrupted (zeroed) by *dinit*(1M) *-g*. Files and directories whose data blocks match an entry in *file* are listed (by name) as CORRUPT. (The list of corrupt blocks may be generated by *fblkgen*(1M).)

If no *file-systems* are specified, *fsck* will read a list of default file systems from the file **/etc/checklist**.

Inconsistencies checked are:

1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Incorrect number of blocks.
   Directory size not 16-byte aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated i-node.
   I-node number out of range.
8. Super Block checks:
   More than 65536 i-nodes.
   More blocks for i-nodes than there are in the file system.
9. Bad free block list format.
   Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the user's concurrence, reconnected by placing them in the **lost+found** directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. Empty files or directories are removed, as long as the **–n** option is not specified. *fsck* will force the reconnection of nonempty directories. The name assigned is the i-node 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 done by making **lost+found**, copying a number of files to the directory, then removing them (before *fsck* is executed).

Checking the raw device is almost always faster and should be used with everything but the root file system.

dfsck

This version of the *fsck* command is only appropriate for computers equipped with dual hard disk drives. *dfsck* should not be used to check the *root* file system.

dfsck allows two file system checks on two different drives simultaneously. *options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A – is the separator between the file system groups.

The *dfsck* program permits a user to interact with two *fsck* programs at once. To aid in this, *dfsck* will print the file system name for each message to the user. When answering a question from *dfsck*, the user must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

FILES

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

SEE ALSO

checkfsys(1M), crash(1M), fsdb(1M), mkfs(1M), ncheck(1M)
uadmin(2), checklist(4), fs(4) in the *Programmer's Reference Manual*.

BUGS

I-node numbers for **.** and **..** in each directory are not checked for validity.

NAME

fsdb – file system debugger

SYNOPSIS

/etc/fsdb *special* [–]

DESCRIPTION

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

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

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

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

The symbols recognized by *fsdb* are:

#	absolute address
i	convert from i-number to i-node address
b	convert to block address
d	directory slot offset
+ , –	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=–	decremental assignment
="	character string assignment
O	error checking flip flop

p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

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

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

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

Dots, tabs, and spaces 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 i-node, 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 address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. I-nodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a#	data block numbers (0 – 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i

prints i-number 386 in an i-node format. This now becomes the current working i-node.

ln=4

changes the link count for the working i-node to 4.

ln=+1

increments the link count by 1.

fc

prints, in ASCII, block zero of the file associated with the working i-node.

2i.fd

prints the first 32 directory entries for the root i-node of this file system.

d5i.fc
: changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.

512B.p0o
: prints the superblock of this file system in octal.

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 when used with **nm** if the first character is alphabetic.

a2b.p0d
: prints the third block of the current i-node as directory entries.

SEE ALSO

fsck(1M), ncheck(1M)
dir(4), fs(4) in the *Programmer's Reference Manual*.

NAME

fsinfo – print various file system information

SYNOPSIS

/usr/lbin/fsinfo [-f] [-i] [-s] [-l] *filesystem* ...

DESCRIPTION

fsinfo is a utility for obtaining information about a file system.

Since non-root users are not able to read file system superblocks, only root may run *fsinfo* successfully.

The following options are available for the *fsinfo* program; at least *one* option must be provided:

-f The -f option prints the number of 512-byte logical blocks currently in use on the specified file systems.

-i The -i option prints the total number of inodes (except for the two used for . and ..) on the specified file systems.

-l The -l option prints the number of 512-byte logical blocks currently free on the specified file systems.

-s The -s option returns zero if the status of the specified file system is **FsOKAY;** nonzero if the status is not FsOKAY. If the file system is currently mounted, a nonzero value is returned.

FILES

/usr/lbin/fsinfo
/usr/include/sys/filsys.h

DIAGNOSTICS

With the **-s**, **-i**, and **-l** options, *fsinfo* returns the number of errors encountered trying to print the requested information. These errors include a file system path that is non-existent or cannot be opened, and file systems whose type (1K, 2K, 4K, 8K) cannot be determined.

fsinfo returns 0 if it was able to access all the specified file systems and provide all the requested information.

For the **-s** option, *fsinfo* returns 0 if the status of the file system is FsOKAY or 1 if the file system is mounted or if its status is not FsOKAY.

NOTES

A comment in the source code indicates that the **-f** option should return the 'free block count', but the code is obviously set up to return the number of blocks 'in use' by calculating total blocks minus free blocks. This may indicate that *fsinfo* implementation will be different on other vendor's systems.

NAME

fsstat – report file system status

SYNOPSIS

/etc/fsstat *special_file*

DESCRIPTION

fsstat reports on the status of the file system on *special_file*. During startup, this command is used to determine if the file system needs checking before it is mounted. *fsstat* succeeds if the file system is unmounted and appears okay. For the root file system, it succeeds if the file system is active and not marked bad.

SEE ALSO

fs(4) in the *Programmer's Reference Manual*.

DIAGNOSTICS

The command has the following exit codes:

0	the file system is not mounted and appears okay, (except for root where 0 means mounted and okay).
1	the file system is not mounted and needs to be checked.
2	the file system is mounted.
3	the command failed.

NAME

fstyp – determine file system identifier

SYNOPSIS

/etc/fstyp *special*

DESCRIPTION

fstyp lets the user determine the file system identifier of mounted or unmounted file systems using heuristic programs. The file system type is required by *mount*(2) and sometimes by *mount*(1M) to mount file systems of different types.

The directory **/etc/fstyp.d** contains a program for each file system type to be checked; each of these programs applies some appropriate heuristic to determine whether the supplied *special* file is of the type for which it checks. If it is, the program prints on standard output the usual file-system identifier for that type and exits with a return code of 0; otherwise, it prints error messages on standard error and exits with a non-zero return code. *fstyp* runs the programs in **/etc/fstyp.d** in alphabetical order, passing *special* as an argument; if any program succeeds, its file-system type identifier is printed and *fstyp* exits immediately. If no program succeeds, *fstyp* prints `Unknown_fstyp` to indicate failure.

WARNING

The use of heuristics implies that the result of *fstyp* is not guaranteed to be accurate.

SEE ALSO

mount(1M)
mount(2), sysfs(2) in the *Programmer's Reference Manual*.

NAME

fumount – forced unmount of an advertised resource

SYNOPSIS

fumount [-w *sec*] *resource*

DESCRIPTION

fumount unadvertises *resource* and disconnects remote access to the resource. The *-w sec* causes a delay of *sec* seconds prior to the execution of the disconnect. When the forced unmount occurs, an administrative shell script is started on each remote computer that has the resource mounted (**/usr/bin/rfuadmin**). If a grace period of seconds is specified, *rfuadmin* is started with the *fuwarn* option. When the actual forced unmount is ready to occur, *rfuadmin* is started with the *fumount* option. See the *rfuadmin*(1M) man page for information on the action taken in response to the forced unmount.

This command is restricted to the superuser.

ERRORS

If *resource* (1) does not physically reside on the local machine, (2) is an invalid resource name, (3) is not currently advertised and is not remotely mounted, or (4) the command is not run with superuser privileges, an error message is sent to standard error.

SEE ALSO

adv(1M), mount(1M), rfuadmin(1M), rfudaemon(1M), rmount(1M), unadv(1M)

NAME

fusage – disk access profiler

SYNOPSIS

fusage [[*mount_point*] | [*advertised_resource*] | [*block_special_device*] [...]]

DESCRIPTION

When used with no options, *fusage* reports block i/o transfers, in kilobytes, to and from all locally mounted file systems and advertised RFS resources on a per client basis. The count data are cumulative since the time of the mount. When used with an option, *fusage* reports on the named file system, advertised resource, or block special device.

The report includes one section for each file system and advertised resource and has one entry for each machine that has the directory remotely mounted, ordered by decreasing usage. Sections are ordered by device name; advertised resources that are not complete file systems will immediately follow the sections for the file systems they are in.

SEE ALSO

adv(1M), mount(1M), df(1M), crash(1M)

NAME

fuser – identify processes using a file or file structure

SYNOPSIS

/etc/fuser [**–ku**] *files* | *resources* [**–**] [[**–ku**] *files* | *resources*]

DESCRIPTION

fuser outputs the process IDs (PIDs) of the processes that are using the *files* or remote *resources* specified as arguments. Each PID is followed by a letter code, interpreted as follows: if the process is using the file as 1) its current directory, the code is **c**, 2) the parent of its current directory (only when the file is being used by the system), the code is **p**, or 3) its **root** directory, the code is **r**.

When the process is executing the file, the code is **t**, when the process is sleeping on a receive descriptor (RFS), the code is **s**, and when it is a server, the code is **S**. For block special devices with mounted file systems, all processes using any file on that device are listed. For remote resource names, all processes using any file associated with that remote resource (RFS) are reported. (*fuser* cannot use the mount point of the remote resource; it must use the resource name.) For all other types of files (text files, executables, directories, devices) only the processes using that file are reported.

The following options may be used with *fuser*:

–u
the user login name, in parentheses, also follows the PID.

–k
the SIGKILL signal is sent to each process. Since this option spawns kills for each process, the kill messages may not show up immediately (see *kill*(2)).

If more than one group of files are specified, the options may be specified again for each additional group of files. A lone dash cancels the options currently in force; then, the new set of options applies to the next group of files.

The PIDs 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.

You cannot list processes using a particular file from a remote resource mounted on your machine. You can only use the resource name as an argument.

Any user with permission to read **/dev/kmem** and **/dev/mem** can use *fuser*. Only the superuser can terminate another user's process.

FILES

/unix	for system namelist
/dev/kmem	for system image
/dev/mem	also for system image

SEE ALSO

mount(1M)
ps(1) in the *User's Reference Manual*.
kill(2), signal(2) in the *Programmer's Reference Manual*.

NAME

fwtmp, wtmpfix – manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [**–ic**]
/usr/lib/acct/wtmpfix [*files*]

DESCRIPTION

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** denotes that input is in ASCII form, and to write output in binary form.

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, *acctcon*(1M) faults when it encounters certain date-change records.

Each time the date is set, a pair of date change records are 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.

In addition to correcting time/date stamps, *wtmpfix* checks the validity of the name field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it changes the login name to **INVALID** and writes a diagnostic to the standard error. In this way, *wtmpfix* reduces the chance that *acctcon*(1) will fail when processing connect accounting records.

You should include the **/usr/lib/acct** in the PATH variable.

EXAMPLES

```
fwtmp </etc/wtmp >/etc/wtmp.ascii
fwtmp -ic </etc/wtmp.ascii >/etc/wtmp
wtmpfix /etc/wtmp >/etc/wtmp.fixed
```

FILES

/etc/wtmp
/usr/include/utmp.h

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), runacct(1M)
acctcom(1), ed(1) in the *User's Reference Manual.*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual.*

NAME

getdevname – find device filename from major and minor numbers

SYNOPSIS

/etc/getdevname major minor

DESCRIPTION

The *getdevname* utility is used to find a device filename that corresponds to a major and minor number.

It reads standard input to obtain a list of pathnames from which it tries to find the first one that corresponds to the arguments major and minor numbers.

If a corresponding filename is found, it is output to standard output. Otherwise, `No match` is printed.

EXAMPLE

The following command prints **/dev/80s0** (if that file exists):

```
/bin/find /dev -type b -print | /etc/getdevname 7 0
```

DIAGNOSTICS

getdevname returns 0 on success; -1 in all other cases

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 system. It can only be executed by the superuser; i.e., a process with the user-ID of **root**. Initially *getty* prints 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. It does this by using the options and arguments specified.

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. Unless *getty* is invoked with the **–h** flag, *getty* forces a hangup on the line by setting the speed to zero 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.

speed, the optional second argument, 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.

type, the optional third argument, is a character string describing to *getty* what type of terminal is connected to the line in question. *getty* recognizes the following types:

none	default
ds40-1	Dataspeed40/1
tektronix,tek	Tektronix
vt61	DEC vt61
vt100	DEC vt100
hp45	Hewlett-Packard 45
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.

linedisc, the optional fourth argument, 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 causes *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

After the user's name has been typed in, it 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.

Finally, *login* is **exec**d with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which places 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.

FILES

/etc/gettydefs

SEE ALSO

ct(1C), init(1M), tty(7)
login(1) in the *User's Reference Manual*.
ioctl(2), gettydefs(4), inittab(4) in the *Programmer's Reference Manual*.

BUGS

While *getty* understands simple single character quoting conventions, it is not possible to quote certain special control characters used by *getty*. Thus, you cannot login via *getty* and type a **#**, **@**, /, **!**, **_**, backspace, **^U**, **^D**, or **&** as part of your login name or arguments. *getty* uses them to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. They are always interpreted as having their special meaning.

NAME

helpadm – make changes to the Help Facility database

SYNOPSIS

/etc/helpadm

DESCRIPTION

The Help Facility Administration command, *helpadm*, allows system administrators and command developers to define the content of the Help Facility database for specific commands and to monitor use of the Help Facility. The *helpadm* command can only be executed by login root, login bin, or a login that is a member of group bin.

The *helpadm* command prints a menu of three types of Help Facility data which can be modified, and two choices relating to monitoring use of the Help Facility. The five choices are:

1. modify *startup* data
2. add, modify, or delete a *glossary* term
3. add, modify, or delete command data (description, options, examples, and keywords)
4. prevent monitoring use of the Help Facility (login root and login bin only)
5. permit monitoring use of the Help Facility (login root and login bin only)

The user may make one of the above choices by entering its corresponding letter (given in the menu), or may exit to the shell by typing **q** (for "quit").

If one of the first three choices is chosen, the user is prompted for additional information; specifically, which *startup* screen, *glossary* term definition, or command description is to be modified. The user may also be prompted for information to identify whether the changes to the database are additions, modifications, or deletions. If the user is modifying existing data or adding new data, they are prompted to make the appropriate modifications/additions. If the user is deleting a *glossary* term or a command from the database, they must respond affirmatively to the next query for the deletion to be done. In any case, before the user's changes are final, they must respond affirmatively when asked whether they are sure they want their requested database changes to be done.

By default, *helpadm* puts the user into *ed*(1) to make additions/ modifications to database information. If the user wishes to be put into a different editor, then they should set the environment variable **EDITOR** in their environment to the desired editor, and then export **EDITOR** .

If the user chooses to monitor/prevent monitoring use of the Help Facility, the choice made is acted on with no further interaction by the user.

FILES

HELPLOG	**/usr/lib/help/HELPLOG**
helpclean	**/usr/lib/help/helpclean**

SEE ALSO

ed(1), glossary(1), help(1), locate(1), starter(1), usage(1)

WARNINGS

When the system is delivered to a customer, **/etc/profile** exports the environment variable **LOGNAME** . If **/etc/profile** has been changed so that **LOGNAME** is not exported, then the options to monitor/prevent monitoring use of the Help Facility may not work properly.

NAME

id – print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

id outputs the user, group, access-group list IDs, and the corresponding names of the invoking process. If the effective and real IDs are different, both are printed.

SEE ALSO

logname(1) in the *User's Reference Manual*.
getuid(2), and getgroups(2) in the *Programmer's Reference Manual*.

NAME

idload – RFS user and group mapping

SYNOPSIS

idload [**–n**] [**–g** *g_rules*] [**–u** *u_rules*] [*directory*]
idload –k

DESCRIPTION

idload is used on Remote File Sharing server machines to build translation tables for user and group ids. It takes your **/etc/passwd** and **/etc/group** files and produces translation tables for user and group ids from remote machines, according to the rules set down in the *u_rules* and *g_rules* files. If you are mapping by user and group name, you need copies of remote **/etc/passwd** and **/etc/group** files. If no rules files are specified, remote user and group ids are mapped to MAXUID+1 (this is an id number that is one higher than the highest number you could assign on your system.)

By default, the remote password and group files are assumed to reside in **/usr/nserve/auth.info/domain/nodename/[passwd | group]**. The *directory* argument indicates that some directory structure other than **/usr/nserve/auth.info** contains the **domain/nodename passwd** and **group** files. (**nodename** is the name of the computer the files are from and **domain** is the domain that computer is a member of.)

You must run *idload* to put the mapping into place. Global mapping takes effect immediately for machines that have one of your resources currently mounted. Mapping for other specific machines takes effect when each machine mounts one of your resources.

–n
used to do a trial run of the id mapping. No translation table is produced, however, a display of the mapping is output to the terminal (*stdout*).

–k
used to print the idmapping that is currently in use. (Specific mapping for remote machines are not shown until that machine mounts one of your resources.)

–u *u_rules*
The *u_rules* file contains the rules for user id translation. The default rules file is **/usr/nserve/auth.info/uid.rules**.

–g *g_rules*
The *g_rules* file contains the rules for group id translation. The default rules file is **/usr/nserve/auth.info/gid.rules**.

This command is restricted to the superuser.

RULES

The rules files have two types of sections (both optional): **global** and **host**. There can be only one global section, though there can be one host section for each computer you want to map.

The **global** section describes the default conditions for translation for any machines that are not explicitly referenced in a **host** section. If the global section is missing, the default action is to map all remote user and group ids from undefined computers to MAXUID+1. The syntax of the first line of the **global** section is:

global

A **host** section is used for each machine or group of machines that you want to map differently from the global definitions. The syntax of the first line of each **host** section is:

host *name* ...

where *name* is replaced by the full name of a computer (*domain.nodename*).

The format of a rules file is described below (all lines are optional, but must appear in the order shown):

```
global
default local | transparent
exclude remote_id-remote_id | remote_id
map remote_id:local

host domain.nodename [domain.nodename...]
default local | transparent
exclude remote_id-remote_id | remote_id | remote_name
map remote:local | remote | all
```

The following describes these instruction types:

The line:

> default *local* | **transparent**

defines the mode of mapping for remote users that are not specifically mapped in instructions in other lines. **transparent** means that each remote user and group id will have the same numeric value locally unless it appears in the **exclude** instruction. *local* can be replaced by a local user name or id to map all users into a particular local name or id number. If the default line is omitted, all users that are not specifically mapped are mapped into a "special guest" login id.

The line:

> exclude *remote_id-remote_id* | *remote_id* | *remote_name*

defines remote ids that will be excluded from the **default** mapping. The **exclude** instruction must precede any **map** instructions in a block. You can use a range of id numbers, a single id number, or a single name. (*remote_name* cannot be used in a *global* block.)

The line:

> map *remote:local* | *remote* | **all**

defines the local ids and names that remote ids and names will be mapped into. *remote* is either a remote id number or remote name; *local* is either a local id number or local name. Placing a colon between a *remote* and a *local* will give the value on the left the permissions of the value on the right. A single *remote* name or id will assign the user or group permissions of the same local name or id. **all** is a predefined alias for the set of all user and group ids found in the local **/etc/passwd** and **/etc/group** files. (You cannot map by remote name in **global** blocks.)

NOTE: *idload* always outputs warning messages for **map all** because password files always contain multiple administrative user names with the same id number. The first mapping attempt on the id number will succeed, each subsequent attempts produce a warning.

RFS does not need to be running to use *idload*.

EXIT STATUS

On successful completion, *idload* produces one or more translation tables and return a successful exit status. If *idload* fails, the command returns an exit status of zero and not produce a translation table.

ERRORS

If (1) either rules file cannot be found or opened, (2) there are syntax errors in the rules file, (3) there are semantic errors in the rules file, (4) **host** password or group information could not be found, or (5) the command is not run with superuser privileges, an error message will be sent to standard error. Partial failures will cause a warning message to appear, though the process will continue.

FILES

/etc/passwd
/etc/group
/usr/nserve/auth.info/*domain***/***nodename***/**[user | group]
/usr/nserve/auth.info/uid.rules
/usr/nserve/auth.info/gid.rules

SEE ALSO

mount(1M)
Remote File Sharing chapter of the *System Administrator's Guide* for detailed information on ID mapping.

NAME

igf – software management *package* generation facility

SYNOPSIS

igf [**-vbu**] [**-f** *special_file*] { *table* | - }

DESCRIPTION

igf is a general purpose utility for creating software installation tapes. It provides a flexible mechanism to combine identification information, multiple archives of software, and scripts needed to manipulate software packages (usually installation and removal scripts) on a single tape. Optionally, the tape can be made bootable to run standalone.

Information about what is written to the tape is read from *table* or standard input when the argument is a dash (–). This information includes the package description, name, version number, and entries that describe the number and contents of the archives. The archives are not limited to a specific format and may even be several different formats on a single tape.

If the **–b** option is used to create a bootable tape, *table* also contains information about a bootloader and other files necessary to run in a standalone environment.

The **–f** option is used to specify a tape device other than the default (**/dev/rmt/ctapen**). Since the tape is created with multiple records, *special_device* must be a "no-rewind" tape device (see WARNINGS).

The **–v** option prints messages showing the progress and contents of the tape being made. The **-v** option tells igf to use "UNIX" instead of "V/88" as the volume id.

table is made up of a number of single and/or multiple line entries separated by any number of blank lines. A line is defined as all characters leading up to a newline character; line continuation is not recognized. Lines within a multiple line entry must not be separated by blank lines. A pound sign (**#**) in the first column of a line indicates a comment line, and the line is discarded. Each line of each entry consists of a keyword immediately followed by an equal sign (=) and the text for that line.

table contains several different types of entries. Some of them are optional, while some are used only when creating a bootable tape. In the following descriptions, lines are referred to by their keyword.

DESC
: Optional description of the software package on the tape. This description is limited to 88 characters.

VER
: Optional package version number. This is actually a string of 12 characters and need not be a number. All printable characters, including spaces, are acceptable.

PACK
: Optional package name. The name is usually the one-word name of the package, limited to 14 characters.

BL
: Pathlist to the tape bootloader. It must be a valid **a.out** file that is readable by the current process (only with **–b** option).

OS
: Pathlist to the standalone operating system or boot file. It must be a valid **a.out** file and readable by the current process (only with **–b** option).

RAM
: Ramdisk image. This entry must be a pathlist to a valid file system image (only with **–b** option).

FILES
: Definition of an archive's contents. This entry begins a multiple line entry. It has two arguments separated by white space: source and destination directory. *igf* changes to the source directory before the the archive command is executed. This is useful for archives with relative pathnames. The destination directory will be stored on the tape and used by the tape extraction facility, *ixf*(1M). The other possible lines in this entry are A_DESC, ARC, O_OPT and I_OPT. The FILES line must be the first line in the entry and is the only required line.

A_DESC
: Archive description. This entry is a short description of the contents of this archive, limited to 25 characters. This line is always optional. If included, it is stored in the tape directory entry for this archive.

ARC

Name of the archive utility. This is the archive utility that is used to write the archive specified by the current FILES entry. The name is not a full pathlist. The environment variable **PATH** is parsed to find the instance of the utility. The first match is used, as it is in the shell. The archiver name will also be stored in the tape directory for extracting of this archive.

O_OPT

Archiver output options. These options are used by the archiver to create the archive. If the string **$DEVICE** is in this entry, it is substituted with the current tape device specified by the **-f** option or by **/dev/rmt/ctapen** when **-f** was not specified. This allows the use of different tape device nodes without having to change *table*.

I_OPT

Archiver input options. These options are used by the archiver when files are being extracted from the tape. They are stored in the tape directory. If the string **$DEVICE** is in this entry, it is replaced with the current tape device specified on the command line of *ixf*(1M) when the tape is being read or extracted. This allows for different device names to be used on the source and destination systems without changing the contents of the tape directory which would require remaking the tape.

SCRIPTS

Scripts archive. This entry is similar to the FILES entry except it has only one argument, a source directory. SCRIPTS is a special archive that is used to contain scripts and all files necessary to manipulate a software package (used by **sysadm softwaremgmt**). A SCRIPTS entry is also a multiple line entry which can contain all the lines associated with the FILES entry (A_DESC, ARC, O_OPT, I_OPT).

All other lines in FILES and SCRIPTS entries are optional. If any one of the ARC, O_OPT or I_OPT lines are specified, then all three of these lines are required to prevent a mismatch of archiver options and arguments with the archive utility. If all three lines are not in the entry, they take on the following values: ARC defaults to *cpio*, O_OPT defaults to **–ocB > $DEVICE**, and I_OPT defaults to **–icBdu < $DEVICE**. File names in and below the source directory will be supplied to *cpio*(1) by *find*(1).

The tape generated by *igf* is made up of multiple records. The number of records depends on whether or not the tape is bootable and how many archives are on it.

The first tape record, called **volume id block**, is a header. It contains information about the tape for the software package and for the tape-based bootloader. There is no line in *table* that creates this record but some information from the table is stored here.

If the tape is bootable, the next record is the bootloader. It is stored on the tape as an image of how it will be run, not as an **a.out** file. The bootloader designed to work with *igf* is **/stand/m88k/boots/350ipl** for SYSTEM V/88 platforms. This record is created from the BL line in *table*.

The next record is a tape directory. It contains the following information about all records after the tape directory (where applicable): record number starting at zero for the **volume id block**, archive number starting at one for the first archive, type of record, name of record or archive utility, description, destination directory, and input options. There is no line in *table* that causes this record to be created.

On a bootable tape, the next record is an operating system or standalone kernel. There may be as many of these records on the tape as needed. One record is created for each OS entry in *table*. They are stored on the tape in **a.out** format. The bootloader, *350ipl*, is able to select any one of these files to boot from.

After the operating system(s), there may be a ramdisk image record on a bootable tape. This record is created from the RAM line in *table*. It is an image of a file system from disk.

Following this record are all of the archives. Archive records are created from FILES and SCRIPTS entries in *table*, one record per entry. If there is a SCRIPTS entry, it is always the first archive; otherwise, the archives are in the order in which they appear in *table*. These archives have no special headers and their format is determined by the archiving utility used.

EXAMPLES

Following is an example of a table used to create a bootable installation tape for a SYSTEM V/88 platform..

— Example Table —

```
    #IGF Table for creating m88k distribution tapes

DESC=SYSTEM V/88 Base Operating System

PACK=BOS

VER=FH32.11
```

```
# Tape bootloader
BL=/root/stand/m88k/boots/350ipl

# kernel used for installation
OS=/root/stand/tapeboot

RAM=/dev/mbfs

#Archive entry for the bill-of-materials files
FILES=/root                    /mnt

ARC=cpio
O_OPT=-ocB > $DEVICE </usr/src/build/tape/releaselist
I_OPT=-icBdu < $DEVICE

#Archive entry for the files in root and usr file systems
FILES=/usr/src/build/tape                    /mnt
ARC=cpio
O_OPT=-ocB > $DEVICE <releasefile
I_OPT=-icBdu < $DEVICE
```

In this example, "tapeboot" is the operating system kernel that will be booted by the tape-based bootloader *350ipl* . The file system which will be mounted as a memory-based file system by the bootloader is specified by the block-special device **/dev/dsk/m323_1s1** .

There are two *cpio*(1) archive entries. The first archive is for the basic operating system files, and the second is for the bill-of-materials file. Since *cpio*(1) gets its file list from the standard input, standard input is redirected from the file **/usr/src/build/tape/releaselist**, which contains a list of pathnames of basic operating systems files for the first archive and **/usr/src/build/tape/releaselist.**

The **$DEVICE** string is used so the tape device node name may be changed for the second archive. Notice that the output is being redirected to the tape.

SEE ALSO

ixf (1M), sh (1)

WARNINGS

Although 9-track drives have the capabilities needed by *igf*, only the MVME350 and MVME327 streaming tape drive are supported.

If *special_device* is not a "no-rewind" device, the program execution will proceed normally without a single error message, but the tape will contain only one record, usually the last archive.

If the **–f** option is used, the *special_device* name must correspond to a ''no-rewind'' device. The name for the ''rewind'' device is derived by truncating the last character from the ''no-rewind'' device's name (for example, **/dev/rmt/ctapen** for a no-rewind device and **/dev/rmt/ctape** for the rewind device).

BUGS

The option to have the *table* information read from standard input does not work although it may appear to.

NAME

infocmp – compare or print out terminfo descriptions

SYNOPSIS

infocmp [**–d**] [**–c**] [**–n**] [**–I**] [**–L**] [**–C**] [**–r**] [**–u**] [**–s d|i|l|c**] [**–v**] [**–V**] [**–1**] [**–w** *width*] [**–A** *directory*] [**–B** *directory*] [*termname* ...]

DESCRIPTION

infocmp can be used to compare a binary *terminfo*(4) entry with other terminfo entries, rewrite a *terminfo*(4) description to take advantage of the **use=** terminfo field, or print out a *terminfo*(4) description from the binary file (*term*(4)) in a variety of formats. In all cases, the boolean fields will be printed first, followed by the numeric fields, followed by the string fields.

Default Options

If no options are specified and zero or one *termnames* are specified, the **–I** option will be assumed. If more than one *termname* is specified, the **–d** option will be assumed.

Comparison Options [–d] [–c] [–n]

infocmp compares the *terminfo*(4) description of the first terminal *termname* with each of the descriptions given by the entries for the other terminal's *termnames*. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: **F** for boolean variables, **–1** for integer variables, and **NULL** for string variables.

–d

produce a list of each capability that is different. In this manner, if one has two entries for the same terminal or similar terminals, using *infocmp* will show what is different between the two entries. This is sometimes necessary when more than one person produces an entry for the same terminal and one wants to see what is different between the two.

–c

produce a list of each capability that is common between the two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **–u** option is worth using.

–n

produce a list of each capability that is in neither entry. If no *termnames* are given, the environment variable **TERM** will be used for both of the *termnames*. This can be used as a quick check to see if anything was left out of the description.

Source Listing Options [–I] [–L] [–C] [–r]

The **–I**, **–L**, and **–C** options will produce a source listing for each terminal named.

–I
use the *terminfo*(4) names

–L
use the long C variable name listed in <**term.h**>

–C
use the *termcap* names

–r
when using **–C**, put out all capabilities in *termcap* form

If no *termnames* are given, the environment variable **TERM** will be used for the terminal name.

The source produced by the **–C** option may be used directly as a *termcap* entry, but not all of the parameterized strings may be changed to the *termcap* format. *infocmp* will attempt to convert most of the parameterized information, but that which it does not will be plainly marked in the output and commented out. These should be edited by hand.

All padding information for strings will be collected together and placed at the beginning of the string where *termcap* expects it. Mandatory padding (padding information with a trailing '/') will become optional.

All *termcap* variables no longer supported by *terminfo*(4), but which are derivable from other *terminfo*(4) variables, will be output. Not all *terminfo*(4) capabilities will be translated; only those variables which were part of *termcap* will normally be output. Specifying the **–r** option will take off this restriction, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output, mandatory padding is not supported, and *termcap* strings were not as flexible, it is not always possible to convert a *terminfo*(4) string capability into an equivalent *termcap* format. Not all of these strings will be able to be converted. A subsequent conversion of the *termcap* file back into *terminfo*(4) format will not necessarily reproduce the original *terminfo*(4) source.

Some common *terminfo* parameter sequences, their *termcap* equivalents, and some terminal types which commonly have such sequences, are:

Terminfo	**Termcap**	**Representative Terminals**
%p1%c	%.	adm
%p1%d	%d	hp, ANSI standard, vt100
%p1%'x'%+%c	%+x	concept
%i	%i	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	%>xy	concept
%p2 is printed before %p1	%r	hp

Use= Option [–u]

–u

produce a *terminfo*(4) source description of the first terminal *termname* which is relative to the sum of the descriptions given by the entries for the other terminals *termnames*. It does this by analyzing the differences between the first *termname* and the other *termnames* and producing a description with **use=** fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using *infocmp* will show what can be done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first *termname*, but one of the other *termname* entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries that has this capability gives a different value for the capability than that in the first *termname*.

The order of the other *termname* entries is significant. Since the terminfo compiler **tic**(1M) does a left-to-right scan of the capabilities, specifying two **use=** entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given in. *infocmp* will flag any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability *after* a **use=** entry that contains that capability will cause the second specification to be ignored. Using *infocmp* to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation time, is specifying extra **use=** fields that are superfluous. *infocmp* will flag any other *termname* **use=** fields that were not needed.

Other Options [–s d|i|l|c] [–v] [–V] [–1] [–w width]

–s

sort the fields within each type according to the argument below:

d leave fields in the order that they are stored in the *terminfo* database.

i sort by *terminfo* name.

l sort by the long C variable name.

c sort by the *termcap* name.

If no **–s** option is given, the fields printed out will be sorted alphabetically by the *terminfo* name within each type, except in the case of the **–C** or the **–L** options, which cause the sorting to be done by the *termcap* name or the long C variable name, respectively.

–v

print out tracing information on standard error as the program runs.

–V

print out the version of the program in use on standard error and exit.

–1

cause the fields to printed out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.

–w

change the output to width characters.

Changing Databases [–A directory] [–B directory]

The location of the compiled *terminfo*(4) database is taken from the environment variable **TERMINFO**. If the variable is not defined, or the terminal is not found in that location, the system *terminfo*(4) database, usually in */usr/lib/terminfo*, will be used. The options **–A** and **–B** may be used to override this location. The **–A** option will set **TERMINFO** for the first *termname* and the **–B** option will set **TERMINFO** for the other *termnames*. With this, it is possible to compare descriptions for a terminal with the same name located in two different databases. This is useful for comparing descriptions for the same terminal created by different people. Otherwise the terminals would have to be named differently in the *terminfo*(4) database for a comparison to be made.

FILES

/usr/lib/terminfo/?/* compiled terminal description database

SEE ALSO

tic(1M), curses(3X), term(4), terminfo(4) in the *Programmer's Reference Manual*.

DIAGNOSTICS

malloc is out of space!

There was not enough memory available to process all the terminal descriptions requested. Run *infocmp* several times, each time including a subset of the desired *termnames*.

use= order dependency found:

A value specified in one relative terminal specification was different from that in another relative terminal specification.

'use=*term*' did not add anything to the description.

A relative terminal name did not contribute anything to the final description.

must have at least two terminal names for a comparison to be done.

The **–u**, **–d** and **–c** options require at least two terminal names.

captoinfo(1M) in the *System Administrator's Reference Manual*.

Chapter 10 of the *Programmer's Guide*.

NOTE

The *termcap* database (from earlier releases of UNIX System V) may not be supplied in future releases.

NAME

init, telinit – process control initialization

SYNOPSIS

/etc/init [**0123456SsQqabc**]

DESCRIPTION

Init

init is a general process spawner. Its primary role is to create processes from information stored in the file **/etc/inittab** (see *inittab*(4)).

At any given time, the system is in one of eight possible run levels. A run level is a software configuration of the system under which only a selected group of processes exist. The processes spawned by *init* for each of these run levels is defined in **/etc/inittab**. *init* can be in one of eight run levels, **0–6** and **S** or **s** (run levels **S** and **s** are identical). The run level changes when a privileged user runs **/etc/init**. This user-spawned *init* sends appropriate signals to the original *init* spawned by the operating system when the system was booted, telling it which run level to change to.

The following are the arguments to *init*:

0 shut the machine down so it is safe to remove the power. Have the machine remove power if it can.

1 put the system in single-user mode. Unmount all file systems except root. All user processes are killed except those connected to the console.

2 put the system in multi-user mode. All multi-user environment terminal processes and daemons are spawned. This state is commonly referred to as the multi-user state.

3 start the remote file sharing processes and daemons. Mount and advertise remote resources. Run level **3** extends multi-user mode and is known as the remote-file-sharing state.

4 is available to be defined as an alternative multi-user environment configuration. It is not necessary for system operation and is usually not used.

5 Stop the system and go to the firmware monitor.

6 Stop the system and reboot to the state defined by the initdefault entry in **/etc/inittab**.

a,b,c
process only those **/etc/inittab** entries having the **a**, **b** or **c** run level set. These are pseudo-states, which may be defined to run certain commands, but which do not cause the current run level to change.

Q,q
re-examine **/etc/inittab**.

S,s
enter single-user mode. When this occurs, the terminal which executed this command becomes the system console. This is the only run level that doesn't require the existence of a properly formatted **/etc/inittab** file. If this file does not exist, then by default the only legal run level that *init* can enter is the single-user mode. When the system enters **S** or **s**, all mounted file systems remain mounted and only processes spawned by init are killed.

When a system is booted, *init* is invoked and the following occurs. First, *init* looks in **/etc/inittab** for the `initdefault` entry (see **inittab**(4)). If there is one, *init* uses the run level specified in that entry as the initial run level to enter. If there is no `initdefault` entry in **/etc/inittab**, *init* requests that the user enter a run level from the virtual system console. If an **S** or **s** is entered, *init* goes to the single-user state. In the single-user state the virtual console terminal is assigned to the user's terminal and is opened for reading and writing. The command **/bin/su** is invoked and a message is generated on the physical console saying where the virtual console has been relocated. Use either *init* or *telinit*, to signal *init* to change the run level of the system. Note that if the shell is terminated (via an end-of-file), *init* will only re-initialize to the single-user state if the **/etc/inittab** file does not exist.

If a **0** through **6** is entered, *init* enters the corresponding run level. Note that, on SYSTEM V/88 Computers, the run levels **0**, **1**, **5**, and **6** are reserved states for shutting the system down; the run levels **2**, **3**, and **4** are available as normal operating states.

If this is the first time since power up that *init* has entered a run level other than single-user state, *init* first scans **/etc/inittab** for boot and bootwait entries (see *inittab*(4)). These entries are performed before any other processing of **/etc/inittab** takes place, providing that the run level entered matches that of the entry. 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. *init* then scans **/etc/inittab** and executes all other entries that are to be processed for that run level.

In a multi-user environment, **/etc/inittab** is set up so that *init* will create a *getty* process for each terminal that the administrator sets up to respawn.

To spawn each process in **/etc/inittab,** *init* reads each entry and for each entry that should be respawned, it forks a child process. After it has spawned all of the processes specified by **/etc/inittab,** *init* waits for one of its descendant processes to die, a powerfail signal, or a signal from another *init* or *telinit* process to change the system's run level. When one of these conditions occurs, *init* re-examines **/etc/inittab**. New entries can be added to **/etc/inittab** at any time; however, *init* still waits for one of the above three conditions to occur before re-examining **/etc/inittab**. To get around this, **init Q** or **init q** command wakes *init* to re-examine **/etc/inittab** immediately.

When *init* comes up at boot time and whenever the system changes from the single-user state to another run state, init sets the *ioctl*(2) states of the virtual console to those modes saved in the file **/etc/ioctl.syscon**. This file is written by *init* whenever the single-user state is entered.

When a run level change request is made *init* sends the warning signal (**SIGTERM**) to all processes that are undefined in the target run level. *init* waits 5 seconds before forcibly terminating these processes via the kill signal (**SIGKILL**).

The shell running on each terminal will terminate when the user types an end-of-file or hangs up. When *init* receives a 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 *init* receives a *powerfail* signal (**SIGPWR**) it scans **/etc/inittab** for special entries of the type *powerfail* and *powerwait*. These entries are invoked (if the run levels permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions during the powerdown of the operating system. Note that in the single-user states, **S** and **s**, only *powerfail* and *powerwait* entries are executed.

FILES

/etc/inittab
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/dev/console
/dev/contty

SEE ALSO

getty(1M), shutdown(1M), gettydefs(4), inittab(4), utmp(4), termios(7)
login(1), sh(1), stty(1), who(1) in the *User's Reference Manual*.
kill(2) in the *Programmer's Reference Manual*.

WARNINGS

init and *telinit* can be run only by someone who is super-user.

The **S** or **s** state must not be used indiscriminately in the **/etc/inittab** file. A good rule to follow when modifying this file is to avoid adding this state to any line other than the **initdefault**.

The change to **/etc/gettydefs** described in the **WARNINGS** section of the *gettydefs*(4) manual page will permit terminals to pass 8 bits to the system as long as the system is in multi-user state (run level greater than 1). When the system changes to single-user state, the *getty* is killed and the terminal attributes are lost. To permit a terminal to pass 8 bits to the system in single-user state, after you are in single-user state, type:

```
stty –istrip cs8
```

DIAGNOSTICS

If *init* finds that it is 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 in the entry, and generate an error message on the system console. It will then refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user-spawned *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 **/etc/inittab**.

When attempting to boot the system, failure of *init* to prompt for a new run level may be because the virtual system console is linked to a device other than the physical system console.

NAME

install – install commands

SYNOPSIS

/etc/install [**–c** *dira*] [**–f** *dirb*] [**–i**] [**–n** *dirc*] [**–m** *mode*] [**–u** *user*] [**–g** *group*] [**–o**] [**–s**] *file* [*dirx* . . .]

DESCRIPTION

The *install* command is 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.

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*, 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 except **–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 except **−c** and **−f**.

−m *mode*
The mode of the new file is set to *mode*. Only available to the superuser.

−u *user*
The owner of the new file is set to *user*. Only available to the superuser.

−g *group*
The group id of the new file is set to *group*. Only available to the superuser.

−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 frequently used file such as **/bin/sh** or **/etc/getty**, where the existing file cannot be removed. May be used alone or with any other options except **−c**.

−s
Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

make(1), cpset(1)

NAME

ixf – software management package extraction facility

SYNOPSIS

ixf [**-Vvptdncax**] [**–{l|m}** *list*] [**–s** *dir*] *tape_dev*

DESCRIPTION

ixf is the facility used to read tapes created by *igf*(1M). Depending on the options given, general information is printed about the contents of the tape, or archives are read. *tape_dev* is the device node to read the tape. It must be a "no-rewind" type device.

Options

-V
Print package version. TP 2 **-v** Verbose mode.

-p
Print package name.

-t
Print table of contents. This includes all of the archives on the media. In verbose mode, it also lists any operating system and ramdisk entries.

-d
Print package description.

-n
No rewind. Do not rewind tape before exiting.

-c
Print creation date of package.

-a
All print options. Combines the following options: **–t, –V, –p, –d, –c**.

-x
If a SCRIPTS entry exists, exit with a special value (13 decimal) to indicate this.

-l *list*
Extract (or give a table of contents if **–t** option) only the archives in *list*. *list* is a comma-separated list of archive numbers. If the IXF_AR_OFFSET environment variable is set, it is added to each archive number in the list to determine which archives to extract (or list in the table of contents).

-m *list*
Select a list of archives to extract. Before any extraction occurs, the user has a chance to interactively change the input options and destination directory of the selected archives. *list* is a comma-separated list of archive numbers. If the IXF_AR_OFFSET environment variable is set, it is added to each archive number in the list to determine which archives to extract.

-s *dir*
Override the default destination directory for a SCRIPTS entry. Use *dir* instead.

ixf has two operation modes. The first mode prints information about the contents of the tape. There are two types of information, general and table of contents. The general information may be printed with any combination of the following options: **–p**,**–d**,**–V**, **–c**. A table of contents may be printed using the **–t** option. The **–a** option combines both types. The second mode is for extracting files from archives. A combination of these two modes is not allowed.

General information about the contents of the tape is always printed to the standard output path. This information may include the package name, description, version number, and media creation date. Each piece of information is printed on its own line. In verbose mode, each line is prepended with a description of that line's contents. For example, **ixf -vp device** may print, "Package Name - xyz". Without the **–v** option, it may print, "xyz".

The table of contents prints a list of the archives on the tape. If the verbose flag is set, the operating system and ramdisk entries are also printed.

The **Archive Number** (or **Arc Num**) is a unique file number assigned to each archive, starting with one for the first archive (which may be a SCRIPTS entry).

The **Tape File Number** is the actual record number of that file on the tape, starting at zero for the **volume id**.

The **Name** is the original file name of the operating system or ramdisk when the tape was made. This is also the name that must be passed to the bootloader, **ipl130**, to boot from an operating system other than the first OS on the tape.

The **Destination Directory** is the directory pathlist that *ixf* changes to before executing the archive command.

The **Input Command/Description** field is the command that will be run to read the archive.

The optional **Description** field is a small description of the archive's contents enclosed in double quotes.

In extraction mode, *ixf* reads the tape directory, then goes to the first archive. Using the archiver, input options, and destination directory stored in the tape directory, it first changes to the destination directory and then executes the command with the input options. It assumes that if the archive command completes successfully, it will advance the tape one record. To do this, the process must open the "no-rewind" device, perform at least one read, and then close the device. If this is not done, successive commands may read the wrong archive. If the command completes without error, the next archive is read in same fashion until all of the archives are read. However, if the first archive is a SCRIPTS entry, *ixf* changes to the default destination directory **/tmp** (or the directory specified by the **–s** option) and then executes the archive command. In this case, no other archives are processed. Upon completion, the tape is rewound.

SEE ALSO

igf (1M)

NAME

killall – kill all active processes

SYNOPSIS

/etc/killall [*signal*]

DESCRIPTION

killall is used by **/etc/shutdown** to kill all active processes not directly related to the shutdown procedure.

killall terminates all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

killall sends *signal* (see *kill*(1)) to all processes not belonging to the above group of exclusions. If no *signal* is specified, a default of **9** is used.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), shutdown(1M)
kill(1), ps(1) in the *User's Reference Manual*.
signal(2) in the *Programmer's Reference Manual*.

WARNINGS

The *killall* command can be run only by the superuser.

NAME

labelit – provide labels for file systems

SYNOPSIS

/etc/labelit *special* [*fsname volume* [**–n**]]

DESCRIPTION

labelit is used to provide labels for unmounted disk file systems or file systems being copied to tape. The **–n** option provides for initial labeling only (this destroys previous contents). You can abort the command by entering your defined interrupt key (DEL, CTL+C).

With the optional arguments omitted, *labelit* prints current label values.

The *special* name should be the physical disk section (e.g., **/dev/dsk/***cntlr*_**0s2**), or the tape (e.g., **/dev/rmt/***cntlr*_**0m**). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (e.g., **root, u1.**) of the file system.

volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette or tape.

For file systems on disk, *fsname* and *volume* are recorded in the superblock.

SEE ALSO

makefsys(1M)
sh(1) in the *User's Reference Manual.*
fs(4) in the *Programmer's Reference Manual.*

NAME

link, unlink – link and unlink files and directories

SYNOPSIS

/etc/link *file1 file2*
/etc/unlink *file*

DESCRIPTION

The *link* command is used to create a file name that points to another file. Linked files and directories can be removed by the *unlink* command; however, it is strongly recommended that the *rm*(1) and *rmdir*(1) commands be used instead of the *unlink* command.

The only difference between *ln*(1) and *link/unlink* is that the latter do exactly what they are told to do, abandoning all error checking. This is because they directly invoke the *link*(2) and *unlink*(2) system calls.

SEE ALSO

rm(1) in the *User's Reference Manual*.
link(2), unlink(2) in the *Programmer's Reference Manual*.

WARNINGS

These commands can be run only by the superuser.

NAME

lpadmin – configure the LP Print Service

SYNOPSIS

/usr/lib/lpadmin –p *printer* **-F***fault recovery* **-c** *class* **-D** *comment* **-e** *printer* **-f allow:***form-list* **-f deny:***form-list* **-h -i** *interface* **-I** *content-type-list* **-I -M -f***form-name*[**-a**[**-o filebreak**]] **-M -S** *print-wheel* **-m** *model* **-o** *printing-option* **-o nobanner -o banner -r** *class* **-S** *list* **-T** *printer-type* **-u allow:***user-list* **-u deny:***user-list* **-U***dail-info* **-v***device* **-A***alert-type*[**-W***integer*]
/usr/lib/lpadmin –x *dest*
/usr/lib/lpadmin –d [*dest*]
/usr/lib/lpadmin –S *print-wheel* **–A** *alert-type* [**–W** *integer*$_1$] [**–Q** *integer*$_2$]

DESCRIPTION

lpadmin configures the LP Print Service by defining printers and devices. It is used to add and change printers, to remove printers from the service, to set or change the system default destination, and to define alerts for print wheels.

Adding or Changing a Printer

The first form of the *lpadmin* command (**lpadmin –p** *printer options*) is used to configure a new printer or to change the configuration of an existing printer. The following *options* are used, and may appear in any order. For ease of discussion, the printer will be referred to as *P*.

–F *fault-recovery*
Restore the LP Print Service after a printer fault, according to the value of *fault-recovery*:

continue
Continue printing on the top of the page where printing stopped. This requires a filter to wait for the fault to clear before automatically continuing.

beginning
Start printing the request again from the beginning.

wait
Disable printing on the printer and wait for the administrator or a user to enable printing again.

During the wait, the administrator or the user who submitted the stopped print request can issue a change request that specifies where printing should resume. If no change request is made before printing is enabled, printing will resume at the top of the page where stopped, if the filter allows; otherwise, the request will be printed from the beginning.

This option specifies the recovery to be used for any print request that is stopped because of a printer fault.

–c *class*
Insert printer *P* into the specified *class*. *Class* will be created if it does not already exist.

–D *comment*
Save this *comment* for display whenever a user asks for a full description of the printer *P* (see **lpstat**(1)). The LP Print Service does not interpret this comment.

–e *printer*
Copy an existing *printer's* interface program to be the interface program for printer *P*.

–f allow:*form-list*
–f deny:*form-list*
Allows (**–f allow**) or denies (**–f deny**) the forms in *form-list* to be printed on printer *P*.

For each printer, the LP Print Service keeps two lists of forms: an "allow-list" of forms that can be used with the printer, and a "deny-list" of forms that shouldn't be used with the printer. With the **–f allow** option, the forms listed are added to the allow-list and removed from the deny-list. With the **–f deny** option, the forms listed are removed from the allow-list and added to the deny-list.

If the allow-list is not empty, the forms in the list can be used with the printer and all others cannot, regardless of the content of the deny-list. If the allow-list is empty, but the deny-list is not, the forms in the deny-list cannot be used with the printer. All forms can be excluded from a printer by having an empty allow-list and putting the word **any** in the deny-list. All forms can be used on a printer by having an empty deny-list and specifying **any** for the allow-list, provided the printer can handle all the characteristics of the forms.

The LP Print Service uses this information as a set of guidelines for determining where a form can be mounted. Administrators, however, are not restricted from mounting a form on any printer. If mounting a form on a particular printer is in disagreement with the information in the allow-list or deny-list, the administrator is warned but the mount is accepted. Nonetheless, if a user attempts to issue a print or change request for a form and printer combination that is in disagreement with the information, the request is accepted only if the form is currently mounted on the printer. If the form is later unmounted before the request can print, the request is canceled and the user notified by mail.

If an administrator tries to name a form as acceptable for use on a printer that doesn't have the capabilities needed by the form, the command is rejected.

Note the other use of **–f** below.

–h
Indicates that the device associated with *P* is hardwired. This option is assumed when adding a new printer unless the **–l** option is supplied.

–i *interface*
Establishes a new interface program for *P*. *Interface* is the path name of the new program.

–I *content-type-list*
Assigns *P* to handle print requests with content of a type listed in *content-type-list*.

The type **simple** is recognized as the default content-type of files in the UNIX system. Such a data stream contains only printable ASCII characters and the following control characters.

Control Character	Octal Value	Meaning
backspace	10_8	move back to previous column, except at beginning of line
tab	11_8	move to next tab stop
linefeed (newline)	12_8	move to beginning of next line
form feed	14_8	move to beginning of next page
carriage return	15_8	move to beginning of current line

To force the print service to not consider **simple** as a valid type for the printer, give an explicit value (e.g., the printer type) in the *content-type-list*. Vice versa, if you do want **simple** included along with other types, you must include **simple** in the *content-type-list*.

Each printer automatically has its printer type included in the list of content types it will accept.

Except for **simple**, each *content-type* name is freely determined by the administrator. If names given as content types are also printer types, the names are accepted without comment, because the LP Print Service recognizes all printer types as potential content types as well.

–l

Indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*, disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.

–M –f *form-name* **[–a [–o filebreak]]**

Mounts the form *form-name* on *P*. Print requests to be printed with the pre-printed form *form-name* will be printed on *P*. If more than one printer has the form mounted and the user has specified any (with the **–d** option of the **lp** command) as the printer destination, then the print request will be printed on the one that also meets the other needs of the request.

The page length and width, and character and line pitches needed by the form are compared with those allowed for the printer, by checking the capabilities in the *terminfo*(4) database for the type of printer. If the form requires attributes that are not available with the printer, the administrator is warned but the mount is accepted. If the form lists a print wheel as mandatory, but the print wheel mounted on the printer is different, the administrator is also warned but the mount is accepted.

If the **–a** option is given, an alignment pattern is printed, preceded by the same initialization of the physical printer that precedes a normal print request, with one exception: no banner page is printed. Printing is assumed to start at the top of the first page of the form. After the pattern is printed, the administrator can adjust the mounted form in the printer and press return for another alignment pattern (no initialization this time), and can continue printing as many alignment patterns as desired. The administrator can quit the printing alignment patterns by typing 'q'.

If the **–o filebreak** option is given, a formfeed is inserted between each copy of the alignment pattern. By default, the alignment pattern is assumed to correctly fill a form, so no formfeed is added.

A form is "unmounted" by mounting a new form in its place using the **–f** option or the **–f none** option. By default, a new printer has no form mounted.

Note the other use of **–f** above.

–M –S *print-wheel*

Mounts the print wheel *print-wheel* on *P*. Print requests to be printed with *print-wheel* will be printed on *P*. If more than one printer has *print-wheel* mounted and the user has specified any (with the **–d** option of the **lp** command) as the printer destination, then the print request will be printed on the one that also meets the other needs of the request.

If the *print-wheel* is not listed as acceptable for the printer, the administrator is warned but the mount is accepted. If the printer does not take print wheels, the command is rejected.

A print wheel is "unmounted" by mounting a new print wheel in its place or by using the option **–S none**.

By default, a new printer has no special print wheel mounted. Until this is changed, a print request that asks for a specific print wheel will not be printed on *P*.

Note the other uses of the **–S** option described below.

–m *model*

Selects a model interface program, provided with the LP Print Service, for printer *P*.

–o *printing-option*

Each **–o** option in the list below is the default given to an interface program if the option is not taken from a preprinted form description or is not explicitly given by the user submitting a request (see *lp*(1)). The only **–o** options that can have defaults defined are:

length=*scaled-decimal-number*
width=*scaled-decimal-number*
cpi=*scaled-decimal-number*
lpi=*scaled-decimal-number*
stty=*'stty-option-list'*

The term "scaled-decimal-number" refers to a non-negative number used to indicate a unit of size. The type of unit is shown by a "trailing" letter attached to the number. Three types of scaled decimal numbers can be used with the LP Print Service: numbers that show sizes in centimeters (marked with a trailing "c"); numbers that show sizes in inches (marked with a trailing "i"); and numbers that show sizes in units appropriate to use (without a trailing letter), i.e., lines, columns, lines per inch, or characters per inch.

The first four default option values should agree with the capabilities of the type of physical printer, as defined in the *terminfo*(4) database for the printer type. If they do not, the command is rejected.

The *stty-option-list* is not checked for allowed values, but is passed directly to the *stty*(1) program by the standard interface program. Any error messages produced by *stty*(1) when a request is processed (by the standard interface program) are mailed to the user submitting the request.

For each printing option not specified, the defaults for the following attributes are defined in the Terminfo entry for the specified printer type:

length
width
cpi
lpi

The default for **stty** is:

```
stty=9600 cs8 –cstopb –parenb –paroff ixon
      –ixany opost –olcuc –onlcr –ocrnl –onocr
      –onlret –ofill nl0 cr0 tab0 bs0 vt0 ff0
```

You can set any of the **–o** options to the default values (which vary for different types of printers), by typing them without assigned values:

length=
width=
cpi=
lpi=
stty=

–o nobanner
Allows users to submit a print request that asks that no banner page be printed.

–o banner
Forces a banner page to be printed with every print request, even when a user asks for no banner page. This is the default; you must specify **–o nobanner** if you want to allow users to specify **–o nobanner** with the **lp** command.

–r *class*
Removes printer *P* from the specified *class*. If *P* is the last member of the printer class *class*, then *class* will be removed.

–S *list*
Allows the print wheels or aliases for character sets named in *list* to be used with *P*.

If the printer is a type that takes print wheels, then *list* is a comma or space separated list of print wheel names. (Enclose the list with quotes if it contains blanks.) These will be the only print wheels considered mountable on the printer. (You can always force a different print wheel to be mounted, however.) Until the option is used to specify a list, no print wheels will be considered mountable on the printer, and print requests that ask for a particular print wheel with this printer will be rejected.

If the printer is a type that has selectable character sets, then *list* is a comma or separated list of character set name "mappings" or aliases. (Enclose the list with quotes if it contains blanks.) Each "mapping" is of the form:

known-name=*alias*

The *known-name* is: a character set number preceded by **cs**, such as **cs3** for character set three; or a character set name from the Terminfo database **csnm** entry. (See *terminfo*(4) in the *Programmer's Reference Manual*.) If this option is not used to specify a list, only the names already known from the Terminfo database or numbers with a prefix of **cs** will be acceptable for the printer.

If *list* is the word **none**, any existing print wheel list or character set aliases will be removed.

Note the other uses of the **–S** option

–T *printer-type*

Assigns the given *printer-type*, a representation of a physical printer of type *printer-type*. *Printer-type* is used to extract data from *terminfo*(4); this data is used to initialize the printer before printing each user's request. Some filters may also use *printer-type* to convert content for the printer. If this option is not used, the default *printer-type* will be **unknown**; no useful information will be extracted from *terminfo*(4) so each user request will be printed without first initializing the printer. Also, this option must be used if the following are to work: **–o cpi=**, **–o lpi=**, **–o width=**, and **–o length=** options of the **lpadmin** and **lp** commands, and the –S and –f options of the **lpadmin** command.

–u allow:*user-list*
–u deny:*user-list*

Allows (**–u allow**) or denies (**–u deny**) the users in *user-list* access to *P*.

For normal access to each printer the LP Print Service keeps two lists of users: an "allow-list" of people allowed to use the printer, and a "deny-list" of people denied access to the printer. With the **–u allow** option, the users listed are added to the allow-list and removed from the deny-list. With the **–u deny** option, the users listed are removed from the allow-list and added to the deny-list.

If the allow-list is not empty, the users in the list are allowed access to the printer and all others are denied access, regardless of the content of the deny-list. If the allow-list is empty, but the deny-list is not, the users in the deny-list are denied access and all others are allowed. If both lists are empty, all users are allowed access. Access can be denied to all users, except the LP Print Service administrator, by putting **any** in the deny-list. To allow everyone access to *P* and effectively empty both lists, put **any** in the allow-list.

–U *dial-info*

Assign the "dialing" information *dial-info* to the printer. *Dial-info* is used with the *dial*(3) routine to call the printer. Any network connection supported by the BNU will work. *Dial-info* can be either a phone number for a modem connection, or a system name for other kinds of connections. If **–U direct** is given, no dialing takes place because the name **direct** is reserved for a printer that is directly connected. If a system name is given, it is used to search for connection details from the file **/usr/lib/uucp/Systems** or related files. The BNU are required to support this option. By default, **–U direct** is assumed.

–v *device*

Associates a *device* with printer *P*. *Device* is the path name of a file that is writable by *lp*. Note that the same *device* can be associated with more than one printer.

–A *alert-type* [**–W** *integer*]

The **–A** option is used to define an alert-type to inform the administrator when a printer fault is detected, and periodically thereafter, until the printer fault is cleared by the administrator. The *alert-types* are:

mail

Send the alert message via mail (see *mail*(1)) to the administrator who issues this command.

write

Write the message to the terminal on which the administrator is logged in. If the administrator is logged in on several terminals, one is chosen arbitrarily.

quiet

Do not send messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the fault has been cleared and printing resumes, messages will again be sent when another fault occurs with the printer.

none

Do not send messages; any existing alert definition for the printer will be removed. No alert will be sent when the printer faults until a different alert-type (except **quiet**) is used.

shell-command

The *shell-command* is run each time the alert needs to be sent. The shell command should expect the message as standard input. If there are blanks embedded in the command, enclose the command in quotes. Note that the **mail** and **write** values for this option are equivalent to the values **mail** *user-name* and **write** *user-name* respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command *unless* he or she has used the *su*(1) command to change to another user ID. If the *su*(1) command has been used to change the user ID, then the *user-name* for the new ID is used.

list
The type of the alert for the printer fault is displayed on the standard output. No change is made to the alert.

The message sent appears as:

```
The printer printer-name has stopped printing for the reason
given below. Fix the problem and bring the printer back
on line. Printing has stopped, but will be restarted in a
few minutes; issue an enable command if you want to res-
tart sooner. Unless someone issues a change request

lp -i request-id -P ...

to change the page list to print, the current request will
be reprinted from the beginning.

The reason(s) it stopped (multiple reasons indicate
reprinted attempts):

reason
```

The LP Print Service can detect printer faults only through an adequate fast filter and only when the standard interface program or a suitable customized interface program is used. Furthermore, the level of recovery after a fault depends on the capabilities of the filter.

If the *printer-name* is **all**, the alerting defined in this command applies to all existing printers.

If the **–W** option is not used to arrange fault alerting for a printer, the default procedure is to mail one message to the administrator of the printer per fault. Similarly, if *integer* is zero, only one message will be sent per fault. If *integer* is a non-zero number, an alert will be sent every *integer* minute(s).

Restrictions
When creating a new printer, either the **–v** or the **–U** option must be supplied. In addition, only one of the following may be supplied: **–e**, **–i**, or **–m**; if none of these three options is supplied, the model standard is used. The **–h** and **–l** keyletters 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).

Removing a Printer Destination

The **–x** *dest* option removes the destination *dest* from the LP Print Service. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. If *dest* is **all**, all printers and classes are removed. No other *options* are allowed with **–x**.

Setting/Changing the System Default Destination

The **–d** [*dest*] option makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. No other *options* are allowed with **–d**.

Setting an Alert for a Print Wheel

–S *print-wheel* **–A** *alert-type* [**–W** *integer*$_1$] [**–Q** *integer*$_2$]

The **–S** *print-wheel* option is used with the **–A** *alert-type* option to send the alert *alert-type* to the administrator to mount the print wheel when there is one or more jobs queued for it. If this command is not used to arrange alerting for a print wheel, no alert will be sent for the print wheel. The *alert-types* are:

mail
Send the alert message via mail (see *mail*(1)) to the administrator who issues this command.

write
Write the message to the terminal on which the administrator is logged in. If the administrator is logged in on several terminals, one is arbitrarily chosen.

quiet
Do not send messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the *print-wheel* has been mounted and subsequently unmounted, messages will again be sent when the number of print requests again exceeds the threshold.

none
Do not send messages until the **–A** option is given again with a different *alert-type* (other than **quiet**).

shell-command
The *shell-command* is run each time the alert needs to be sent. The shell command should expect the message as standard input. If there are blanks embedded in the command, enclose the command in quotes.

Note that the **mail** and **write** values for this option are equivalent to the values **mail** *user-name* and **write** *user-name* respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command *unless* he or she has used the *su*(1) command to change to another user ID. If the *su*(1) command has been used to change the user ID, then the *user-name* for the new ID is used.

list
The type of the alert for the print wheel is displayed on the standard output. No change is made to the alert.

The message sent appears as:

```
The print wheel print-wheel needs to be mounted
on the printer(s):
printer (integer3 requests)
integer4 print requests await this print wheel.
```

The printers listed are those that the administrator had earlier specified were candidates for this print wheel. The number (*integer*$_3$) listed next to each printer is the number of requests eligible for the printer. The number (*integer*$_4$) shown after the printer list is the total number of requests awaiting the print wheel. It will be less than the sum of the other numbers if some requests can be handled by more than one printer.

If the *print-wheel* is **all**, the alerting defined in this command applies to all print wheels already defined to have an alert.

If the **–W** option is not given or *integer*$_1$ is zero (which is interpreted as **once** and is also the default), only one message will be sent per need to mount a print wheel. If *integer*$_1$ is a non-zero number, an alert will be sent every *integer*$_1$ minute(s).

If the **–Q** option is also given, the alert will be sent when *integer*$_2$ print requests that need the print wheel are waiting. If the **–Q** option is not given, or *integer*$_2$ is 1 or the word **any** (which are both the default), a message is sent as soon as anyone submits a print request for the print wheel when it is not mounted.

The **–S** option has a different meaning when used with the **–p** option.

FILES
/usr/spool/lp/*

SEE ALSO

accept(1M), lpsched(1M)
enable(1), lp(1), lpstat(1), stty(1) in the *User's Reference Manual*.
dial(3), terminfo(4) in the *Programmer's Reference Manual*.

1M)

NAME

lpfilter – administer filters used with the LP Print Service

SYNOPSIS

/usr/lib/lpfilter –f *filter-name* **–F** *path-name*
/usr/lib/lpfilter –f *filter-name* **–**
/usr/lib/lpfilter –f *filter-name* **–i**
/usr/lib/lpfilter –f *filter-name* **–x**
/usr/lib/lpfilter -f *filter*-**all –l**

DESCRIPTION

The *lpfilter* command is used to add, change, delete, and list filters used with the LP Print Service. These filters are used to convert the content type of a file to a content type acceptable to a given printer. One of the following options must be used with the *lpfilter* command: **–F** *path-name* (or – for standard input) to add or change a filter; **–i** to reset an original LP Print Service filter to its factory setting; **–x** to delete a filter; or **–l** to list a filter description.

The argument **all** can be used instead of a *filter-name* with any of these options. When **all** is specified with the **–F** or **–** option, the requested change is made to all filters. Using **all** with the **–i** option has the effect of restoring to their original settings all filters for which predefined settings were initially available. Using the **all** argument with the **–l** option produces a list of all filters, and using it with the **–x** option results in all filters being deleted.

Adding or Changing a Filter

The filter named in the **–f** option and described in the input is added to the filter table. If the filter already exists, its description is changed to reflect the new information in the input. Once added, a filter is available for use.

The filter description is taken from the *path-name* if the **–F** option is given, or from the standard input if the **–** option is given. One of the two must be given to define or change a filter. If the filter named is one originally delivered with the LP Print Service, the **–i** option will restore the original filter description.

When an existing filter is changed with the **–F** or **–** option, items that are not specified in the new information are left as they were. When a new filter is added with this command, unspecified items are given default values.

Note that a filter name and a command must be given. A filter with no input type value is assumed to work with any input type; this is also true for the output type, printer type, and printer values.

Filters are used to convert the content of a request into a data stream acceptable to a printer. For a given print request, the LP Print Service will know the following: the type of content in the request, the name of the printer, the type of the printer, the types of content acceptable to the printer, and the modes of printing asked for by the originator of the request. It will use this information to find a filter that will convert the content into a type acceptable to the printer.

Below is a list of items that provide input to this command, and a description of each item. All lists are comma or space separated.

Input types: *content-type-list*
Output types: *content-type-list*
Printer types: *printer-type-list*
Printers: *printer-list*
Filter type: *filter-type*
Command: *shell-command*
Options: *template-list*

Input types
This gives the types of content that can be accepted by the filter.

Output types
This gives the types of content that the filter can produce from any of the input content types.

Printer types
This gives the type of printers for which the filter can be used. The LP Print Service will restrict the use of the filter to these types of printers.

Printers
This gives the names of the printers for which the filter can be used. The LP Print Service restricts the use of the filter to just the printers named.

Filter type

This marks the filter as a "slow" filter or a "fast" filter. Slow filters are generally those that take a long time to convert their input. They are run unconnected to a printer, to keep the printers from being tied up while the filter is running. Fast filters are generally those that convert their input quickly, or those that must be connected to the printer when run. These will be given to the interface program to run connected to the physical printer.

Command

This specifies the program to run to invoke the filter. The program name as well as fixed options are included in the *shell-command*; additional options are constructed, based on the characteristics of each print request and on the **Options** field.

The command must accept a data stream as standard input and produce the converted data stream on its standard output. This allows filter pipelines to be constructed to convert data not handled by a single filter.

Options

This is a comma separated list of templates used by the LP Print Service to construct options to the filter from the characteristics of each print request listed in the table later.

In general, each template is of the following form:

keyword pattern = *replacement*

The *keyword* names the characteristic that the template attempts to map into a filter specific option; each valid *keyword* is listed in the table below. A *pattern* is either a literal pattern of one of the forms listed in the table, or a single asterisk (*); if the *pattern* matches the value of the characteristic, the template fits and is used to generate a filter specific option. A *pattern* of * matches any value. The *replacement* is a string used as a filter specific option; if an asterisk (*) is used, it is replaced with the value of the characteristic.

lp Option	Characteristic	*keyword*	Possible*patterns*
–T	Content type (input)	INPUT	content-type
N/A	Content type (output)	OUTPUT	content-type
N/A	Printer type	TERM	printer-type
–f, –o cpi=	Character pitch	CPI	integer
–f, –o lpi=	Line pitch	LPI	integer
–f, –o length=	Page length	LENGTH	integer
–f, –o width=	Page width	WIDTH	integer
–P	Pages to print	PAGES	page-list
–S	Character set/ print wheel	CHARSET	character-set-name/ print-wheel-name
–f	Form name	FORM	form-name
–y	Modes	MODES	mode
–n	Number of copies	COPIES	*integer*

For example, the template:

MODES landscape = –l

would show that if a print request was submitted with the **–y landscape** option, the filter should be given the option **–l**. As another example, the template:

TERM * = –T *

would show that the filter should be given the option **–T** *printer-type* for whichever *printer-type* is associated with a print request using the filter.

Deleting a Filter

The **–x** option is used to delete the filter specified in *filter-name* from the LP filter table.

Listing a Filter Description

The **–l** option is used to list the description of the filter named in *filter-name*. If the command is successful, the following message is sent to standard output:

```
Input types: content-type-list
Output types: content-type-list
```

`Printer types`: *printer-type-list*
`Printers`: *printer-list*
`Filter type`: *filter-type*
`Command`: *shell-command*
`Options`: *template-list*

If the command fails, an error message is sent to standard error.

SEE ALSO

lpadmin(1M)
lp(1) in the *User's Reference Manual*

M)

NAME

lpforms – administer forms used with the LP print service

SYNOPSIS

/usr/lib/lpforms –f *form-name options*

/usr/lib/lpforms –f *form-name* **–A** *alert-type* [–Q *integer*$_1$] [**–W** *integer sub* 2]

/usr/lib/lpforms –f *form-name* **–A list**

/usr/lib/lpforms –f *form-name* **–A quiet**

/usr/lib/lpforms –f *form-name* **–A none**

DESCRIPTION

The *lpforms* command is used to administer the use of preprinted forms, such as company letterhead paper, with the LP print service. A form is specified by the *form-name* given with the *lpforms* command. Users may request a form by *form-name* (see *lp*(1)). The argument **all** can be used instead of *form-name* with any of the five *lpforms* command lines shown above. The first command line allows the administrator to add, change, and delete forms, to list the attributes of an existing form, and to allow and deny users access to particular forms. The second command line is used to establish the method by which the administrator is alerted that a form must be mounted on a printer. By using the third command line, an administrator can list the current alerting methods assigned to forms. The fourth command *lpforms* command line is used to terminate an active alert, and the fifth command line is used to remove an alert definition.

With the first *lpforms* command line, one of the following options must be used:

–F *path-name*
  to add or change a form as specified by the information in *path-name*

– to add or change a form, and supply information from standard input

–x
  to delete a form (this option must be used separately; it cannot be used with any other option)

–l
  to list the attributes of a form

–u allow:*user-list*
  to allow users to request a form (this option can be used with the **–F** or – option)

–u deny:*user-list*
to deny users access to a form (this option can be used with the **–F** or **–** option)

Each option is explained below.

Adding or Changing a Form

The **–F** *path-name* option is used to add a new form to the LP Print Service, or to change the attributes of an existing form. The form description is taken from *path-name* if the **–F** option is given, or the standard input if the **–** option is used. One of the two options must be used to define or change a form. *path-name* is the path name of a file that contains all or any subset of the following information about the form.

Page length: $scaled-decimal-number_1$
Page width: $scaled-decimal-number_2$
Number of pages: *integer*
Line pitch: $scaled-decimal-number_3$

Character pitch: $scaled-decimal-number_4$
Character set choice: *character-set/print-wheel*,[**mandatory**]
Ribbon color: *ribbon-color*
Comment:
comment
Alignment pattern: [*content-type*]
content

Except for the last two lines, the above lines can appear in any order. The **Comment:** and *comment* items must appear in consecutive order but can appear before the other items, and the **Alignment pattern:** and the *content* items must appear in consecutive order at the end of the file. Also, the *comment* item cannot contain a line that begins with any of the key phrases above, unless the key phrase is preceded with a > sign. Any leading > sign found in the *comment* will be removed when the comment is displayed. Case distinctions in the key phrases are ignored.

When this command is issued, the form specified by *form-name* is added to the list of forms. If the form already exists, its description is changed to reflect the new information in the input. Once added, a form is available for use in a print request, except where access to the form has been restricted, as described under the **–u allow:** option. A form may also be allowed to be used on certain printers only.

A description of each form attribute follows:

Page length and Page Width

Before printing the content of a print request needing this form, the generic interface program provided with the LP Print Service initializes the physical printer to handle pages *scaled−decimal−number*$_1$ long, and *scaled−decimal−number*$_2$ wide using the printer type as a key into the *terminfo*(4) database. A *scaled-decimal-number* is an optionally scaled decimal number that gives a size in lines, columns, inches, or centimeters, as appropriate. The scale is indicated by appending the letter "i" for inches, or the letter "c" for centimeters. For length or width settings, an unscaled number indicates lines or columns; for line pitch or character pitch settings, an unscaled number indicates lines per inch or characters per inch (the same as a number scaled with "i"). For example, **length=66** indicates a page length of 66 lines, **length=11i** indicates a page length of 11 inches, and **length=27.94c** indicates a page length of 27.94 centimeters.

The page length and page width will also be passed, if possible, to each filter used in a request needing this form.

Number of pages

Each time the alignment pattern is printed, the LP Print Service attempts to truncate the *content* to a single form by, if possible, passing to each filter the page subset of 1–*integer*.

Line pitch and
Character pitch

Before printing the content of a print request needing this form, the interface programs provided with the LP Print Service initializes the physical printer to handle these pitches, using the printer type as a key into the *terminfo*(4) database. Also, the pitches will be passed, if possible, to each filter used in a request needing this form. *Scaled-decimal-number*$_3$ is in lines per centimeter if a ''c'' is appended, and lines per inch otherwise; similarly, *scaled−decimal−number*$_4$ is in columns per centimeter if a ''c'' is appended, and columns per inch otherwise. The character pitch can also be given as **elite** (12 characters per inch), **pica** (10 characters per inch), or **compressed** (as many characters per inch as possible).

Character set choice

When the LP Print Service alerts an administrator to mount this form, it will also mention that the print wheel *print-wheel* should be used on those printers that take print wheels. If printing with this form is to be done on a printer that has selectable or loadable character sets instead of print wheels, the interface programs provided with the LP Print Service will automatically select or load the correct character set. If **mandatory** is appended, a user is not allowed to select a different character set for use with the form; otherwise, the character set or print wheel named is a suggestion and a default only.

Ribbon color

When the LP print service alerts an administrator to mount this form, it will also mention that the color of the ribbon should be *ribbon-color*.

Comment

The LP Print Service displays the *comment* unaltered when a user asks about this form (see *lpstat*(1)).

Alignment pattern

When mounting this form an administrator can ask for the *content* to be printed repeatedly, as an aid in correctly positioning the preprinted form. The optional *content-type* defines the type of printer for which *content* had been generated. If *content-type* is not given, **simple** is assumed. Note that the *content* is stored as given, and will be readable only by the user **lp**.

When an existing form is changed with this command, items missing in the new information are left as they were. When a new form is added with this command, missing items will get the following defaults:

Page Length: **66**
Page Width: **80**
Number of Pages: **1**
Line Pitch: **6**
Character Pitch: **10**
Character Set Choice: **any**
Ribbon Color: **any**
Comment: (no default)
Alignment Pattern: (no default)

Deleting a Form

The **–x** option is used to delete the form specified in *form-name* from the LP Print Service.

Listing Form Attributes

The **–l** option is used to list the attributes of the existing form specified by *form-name*. The attributes listed are those described under *Adding and Changing a Form*, above. Because of the potentially sensitive nature of the alignment pattern, only the administrator can examine the form with this command. Other people can use the *lpstat*(1) command to examine the non-sensitive part of the form description.

Allowing and Denying Access to a Form

The LP Print Service keeps two lists of users for each form: an "allow-list" of people allowed to use the form, and a "deny-list" of people denied access to the form. With the **–u allow:** option, the users listed are added to the allow-list and removed from the deny-list. With the **–u deny:** option, the users listed are removed from the allow-list and added to the deny-list.

If the allow-list is not empty, the users in the list are allowed access to the form and all others are denied access, regardless of the content of the deny-list. If the allow-list is empty, but the deny-list is not, the users in the deny-list are denied access and all others are allowed. If both lists are empty, all users are allowed access. Access can be denied to all users, except the LP Print Service administrator, by putting **any** in the deny-list. To effectively empty both lists, allowing access for everyone, put **any** in the allow-list.

Alerting to Mount Forms

The second *lpforms* command line (shown under **Synopsis**) is used to arrange for alerts to be sent to the administrator when forms need to be mounted on a printer.

When *integer*$_1$ print requests needing the preprinted form *form-name* become queued up because no printer satisfying all the needs of the requests has the form mounted (and for as long as this condition remains), an alert is sent to the administrator every *integer*$_2$ minutes until the form is mounted on a qualifying printer. If the *form-name* is **all**, the alerting defined in this command applies to all existing forms. No alerting is done for a backlog of print requests needing a form if the administrator does not use this command.

The method by which the alert is sent depends on the value of the **–A** option.

write
The message is sent via *write(1)* to the terminal on which the administrator is logged in when the alert arises. If the administrator is logged in on several terminals, one is arbitrarily chosen.

mail
The message is sent via *mail*(1) to the administrator who issues this command.

The message sent appears as:

> **The form** *form-name* **needs to be mounted on the printer(s)** *printer (integer*$_5$ **requests).**
> *integer sub 4* **print requests await this form.**
> **Use the** *ribbon-color* **ribbon.**
> **Use the** *print-wheel* **print wheel, if appropriate.**

The printers listed are those that the administrator had earlier specified were candidates for this form. The number (*integer*$_3$) listed next to each printer is the number of requests eligible for the printer. The number (*integer*$_4$) shown after the list of printers is the total number of requests awaiting the form. It will be less than the sum of the other numbers if some requests can be handled by more than one printer. The *ribbon-color* and *print-wheel* are those specified in the form description. The last line in the message is always sent, even if none of the printers listed use print wheels, because the administrator may choose to mount the form on a printer that does use a print wheel.

Where any color ribbon or any print wheel can be used, the statements above will read:

```
Use any ribbon.
Use any print-wheel.
```

shell-command

The *shell-command* is run each time the alert needs to be sent. The shell command should expect the message as standard input. Note that the **mail** and **write** values for the **–A** command are equivalent to the values "**mail** *user-name*" and "**write** *user-name*," respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command *unless* he or she has used the *su* command to change to another user ID. If the *su* command has been used to change the user ID, then the *user-name* for the new ID is used.

If the **–Q** option is not given or *integer*$_1$ is one or the word **any** (which are both the default), a message is sent as soon as anyone submits a print request for the form when it is not mounted.

If the **–W** option is not given or *integer*$_2$ is zero or the word **once** (which are both the default), only one message is sent when the queue size exceeds *integer*$_1$. If *integer*$_2$ is a non-zero number, an alert will be sent every *integer*$_2$ minute(s).

Listing the Current Alert

The third *lpforms* command line (shown under **Synopsis**) is used to list the type of the alert for the specified form. No change is made to the alert. If *form-name* is recognized by the LP Print Service, one of the following lines is sent to the standard output, depending on the type of alert for the form.

- `When` *integer sub 1* `are queued: alert with "`*shell-command*`" every integer sub 2 minutes`
- `When` *integer sub 1* `are queued: write to` *user-name* `every` *integer sub 2* `minutes`
- When *integer sub 1* `are queued: mail to` *user-name* `every` *integer sub 2* `minutes`
- No alert

The phrase `every` *integer* `minutes` is replaced with `once` if *integer sub 2* (–W *integer sub* 2) is 0.

Terminating an Active Alert

The **–A quiet** option is used to stop messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the form has been mounted and then unmounted, messages will again be sent when the queue size reaches *integer*$_1$ pending requests.

Removing an Alert Definition

No messages will be sent after the **–A none** option is used until the **–A** option is given again with a different *alert-type*. This can be used to permanently stop further messages from being sent as any existing alert definition for the form will be removed.

SEE ALSO

lpadmin(1M), terminfo(4)
lp(1) in the *User's Reference Manual.*

NAME

lpsched, lpshut, lpmove – start/stop the LP Print Service and move requests

SYNOPSIS

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove *requests dest*
/usr/lib/lpmove *$dest_1$ $dest_2$*

DESCRIPTION

lpsched starts the LP print service; this can be done only by **root** or **lp**.

lpshut shuts down the print service. All printers that are printing at the time *lpshut* is invoked does stop printing. When *lpsched* is started again, requests that were printing at the time a printer was shut down will be reprinted from the beginning.

lpmove moves requests that were queued by *lp*(1) between LP destinations. The first form of the *lpmove* command shown above (under *Synopsis*) moves the named *requests* to the LP destination *dest*. *Requests* are request-ids as returned by *lp*(1). The second form of the *lpmove* command moves all requests for destination $dest_1$ to destination $dest_2$; *lp*(1) will then reject any new requests for $dest_1$.

Note that when moving requests, *lpmove* never checks the acceptance status (see *accept*(1M)) of the new destination. Also, the request-ids of the moved request are not changed, so that users can still find their requests. The *lpmove* command does not move requests that have options (content type, form required) that cannot be handled by the new destination.

If a request was originally queued for a class or the special destination **any**, its destination is changed to *new-destination*. A request thus affected is printable only on *new-destination* and not on other members of the **class** or other acceptable printers if the original destination was **any**.

NOTE

By default, the directory **/usr/spool/lp** is used to hold all the files used by the LP Print Service. This can be changed by setting the **SPOOLDIR** environment variable to another directory before running *lpsched.* If you do this, you should populate the directory with the same files and directories found under **/usr/spool/lp**; the LP print service will not automatically create them. Also, the **SPOOLDIR** variable must then be set before any of the other LP Print Service commands are run.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), lpadmin(1M)
enable(1), lp(1), lpstat(1) in the *User's Reference Manual.*

NAME

lpusers – set printing queue priorities

SYNOPSIS

/usr/lib/lpusers –d *priority-level*
/usr/lib/lpusers –q *priority-level* **–u** *user-list*
/usr/lib/lpusers –u *user-list*
/usr/lib/lpusers –q *priority-level*
/usr/lib/lpusers –l

DESCRIPTION

The *lpusers* command is used to set limits to the queue priority level that can be assigned to jobs submitted by users of the LP print service.

The first form of the command (with **–d**) sets the system-wide priority default to *priority-level,* where *priority-level* is a value of 0 to 39, with 0 being the highest priority. If a user does not specify a priority level with a print request (see *lp*(1)), the default priority is used. Initially, the default priority level is 20.

The second form of the command (with **–q** and **–u**) sets the default highest *priority-level* (0-39) that the users in *user-list* can request when submitting a print request. Users that have been given a limit cannot submit a print request with a higher priority level than the one assigned, nor can they change a request already submitted to have a higher priority. Any print requests with priority levels higher than allowed will be given the highest priority allowed.

The third form of the command (with **–u**) removes the users from any explicit priority level, and returns them to the default priority level.

The fourth form of the command (with **–q**) sets the default highest priority level for all users not explicitly covered by the use of the second form of this command.

The last form of the command (with **–l**) lists the default priority level and the priority limits assigned to users.

SEE ALSO

lp(1) in the *User's Reference Manual*.

NAME

m323rd – read disk resident manufacturer's bad track list

SYNOPSIS

m323rd -L *maxcylinder heads maxbads file rawdevice*

m323rd -l *maxcylinder heads maxbads file rawdevice*

DESCRIPTION

The MVME323 devices have a manufacturer's bad track list on the disk in addition to the one written by the *dinit*(1M) utility. This command is used to read the manufacturer's list from the disk and write it to *file*.

The **L** option writes the bad track list in head/cylinder format. The **l** option writes the bad track list in head/cylinder/bfi (bytes from index) format.

maxcylinder
: is the maximum number of cylinders. Typically the last cylinder contains the manufacturer's defect list.

hads
: is the number of surfaces per drive.

maxbads
: is the maximum number of bad spots that are allowable on the disk.

file
: is the name of the bad track file to create after reading the manufacturer's list.

rawdevice
: is the name of the device for the selected drive, for example, **/dev/rdsk/m323_0s7** for drive 0.

SEE ALSO

dinit(1M), ddefs(1M)

NAME

m332xctl – MVME332XT control program

SYNOPSIS

\tc/m332xctl { –t | –r | –R | –D | –h<on|off|info> –s | –g | { –d<dlfile> [–x<sname>] . . . –l | –e<fname>}} dev

DESCRIPTION

m332xctl provides a functional control interface to the MVME332XT Communications Controller. Note that *m332xctl* provides no support for the MVME332 hardware and firmware architecture. The following options and fields are interpreted by *m332xctl:*

–t
Test the existence of the MVME332XT. Return 0 if it exists, else return ENXIO.

–r
Get firmware and driver version and revision numbers The designated **dev** should be the printer device.

–R
Get firmware version number in short format. The designated **dev** should be the printer device.

–D
Debug mode.

–h
Hardware flow control handshaking can be enabled or disabled, and hardware flow control port status can be queried. Hardware flow control is implemented with the RS-232 RTS and CTS handshakes.

–s
Get symbol table of the MVME332XT firmware and display. The designated **dev** should be the printer device.

–g
Get downloadable area information from the MVME332XT controller. The address and size of the download area is displayed. The designated **dev** should be the printer device.

–d
: Download a coff file to the MVME332XT. The designated **dev** should be the printer device.

–x
: Exclude a section when downloading. Up to sixteen sections may be excluded for a particular download operation. This option must be preceded by the **–d** option in the command invocation.

–l
: Instruct the MVME332XT firmware to copy the downloaded line switch table to its internal data structure. This option must be preceded by the **–d** option in the command invocation.

–e
: Instruct the MVME332XT firmware to execute a user function in a downloaded file. This option must be preceded by the **–d** option in the command invocation.

dev
: MVME332XT serial I/O or printer device. **dev** should be the printer device for the **–g**, **–d**, **–r**, **–R**, **–s**, **–l**, and **–x** options.

dlfile
: coff compatible file that is to be linked to the MVME332XT symbol table before downloading.

sname
: Section names to be excluded when **dlfile** is downloaded.

fname
: Function within the **dlfile** that is to be executed.

To obtain the MVME332XT firmware version and revision number, execute the following *m332xctl* command:

m332xctl –R /dev/m332x*XY*

This command issues a message of the form:

VR

where *V* and *R* are the MVME332XT firmware version and revision numbers, respectively.

The *m332xctl* command:

m332xctl –h on /dev/m332x*XY*

enables the hardware flow control option for the specified serial I/O port. The I/O device to be set is designated by the *XY* field throughout this document, where *X* and *Y* refer to the MVME332XT controller and port device numbers, respectively.

Hardware flow control for any MVME332XT serial port may be disabled by issuing:

m332xctl –h off /dev/m332x*XY*.

Hardware flow control is implemented with the RS-232 RTS/CTS signal pairs. In this mode, a serial port transmitter is disabled when its CTS input negates, and a receiver negates its RTS output when the associated receive channel character high water mark has been reached. A MVME332XT serial port hardware flow control configuration may be determined with the following *m332xctl* command.

m332xctl –h info /dev/m332x*XY*

In this example, if hardware flow control is enabled for the specified port, the following message will be sent to standard output:

hardware handshake is enabled

If hardware flow control is disabled for the specified port, the following message will be sent to standard output:

hardware handshake is disabled

To get the start address and size of the MVME332XT download area, use the following *m332xctl* command.

m332xctl –g /dev/m332x*XY*

where **/dev/m332x***XY* must be the MVME332XT printer device. This restricts downloading and download area information access to root.

The following information is displayed in response to the previous command:

Downloadable area start address = *AAAA*, **size** = *SSSS*

The downloadable coff file should be linked to the displayed start address before downloading to the MVME332XT firmware, using the following syntax:

m332xctl –d dlfile /dev/m332x*XY*

where **dlfile** is the coff file to be downloaded, and **/dev/m332x*XY*** is the MVME332XT printer device name, required for security purposes.

To exclude sections of **dlfile** during the download operation, use

m332xctl –d dlfile –x sname1 . . . –x sname*n* /dev/m332x*XY*

where **sname1, . . . , sname*n*** are the section names that are to be excluded during the download operation. The *m332xctl* command supports up to 16 excluded section names using the syntax shown.

The MVME332XT firmware supports user supplied line disciplines via the *m332xctl* **–d** and **–l** options, which allow the downloaded line switch table to be copied to the MVME332XT firmware data structures:

m332xctl –d dlfile –l /dev/m332x*XY*

where **dlfile** is the download file name and **/dev/m332x*XY*** is the MVME332XT printer device special file name, as before. The downloaded **dlfile** must contain the following symbols:

Symbol	Description
– linetable :	**linesw lineswitch table**
– linecount :	**number of lines to be downloaded**

Refer to *mvme332XT(7)* for discussion regarding linesw table structure. Notice that the linesw table structure defined in *mvme332XT(7)* differs from that described in **/usr/include/sys/conf.h**. Intimate familiarity with the MVME332XT firmware architecture is required to successfully port a user developed line discipline.

To download a coff file, **dlfile**, to the MVME332XT and execute a downloaded function, **fname**, use the following syntax:

m332xctl –d dlfile –e fname /dev/m332x*XY*

where **dlfile** is the downloaded file, **fname** is the function to be executed by the MVME332XT firmware, and **/dev/m332x*XY*** is the MVME332XT printer device name. See *mvme332xt(7)* for more information regarding special file naming conventions.

The **–D** option enables the debug mode. Option **–DD** enables the debug mode at level 2, which results in more comprehensive debug messages. Either mode is useful for monitoring a downloading operation and for debugging user developed lineswitch and function routines.

FILES

/dev/m332x*

SEE ALSO

mvme332xt(7), termios(7), tty(7)
stty(1) in the *User's Reference Manual*.
ioctl(2) in the *Programmer's Reference Manual*.

NAME

m336ctl – MVME336 control program

SYNOPSIS

/etc/m336ctl [*romrev* | *on* | *off* | *status*] *dev*

DESCRIPTION

m336ctl provides a functional control interface to the MVME336 Communications Controller. The following options and fields are interpreted by *m336ctl:*

romrev

Get firmware revision number

To obtain the MVME336 firmware version and revision number, execute the following *m336ctl* command:

m336ctl romrev /dev/tty*XY*

This command issues a message of the form:

```
m336ctl: The ROM revision is VR
```

where *V* and *R* are the MVME336 firmware version and revision numbers, respectively.

on off status

Hardware flow control handshaking can be enabled or disabled, and hardware flow control port status can be queried. Hardware flow control is implemented with the RS-232 RTS and CTS handshakes.

The following command enables hardware flow control option for the specified serial I/O port.

m336ctl on /dev/tty*XY*

The I/O device to be set is designated by the *XY* field throughout this document, where *X* is the cluster ID *a* through *f* and *Y* refers to the port device numbers *01* through *16*, respectively.

Hardware flow control for any MVME336 serial port may be disabled by issuing the command:

m336ctl off /dev/tty*XY*.

Hardware flow control is implemented with the RS-232 RTS/CTS signal pairs. In this mode, a serial port transmitter is disabled when its CTS input negates; and a receiver negates its RTS output when the associated receive channel character high water mark has been reached. A MVME336 serial port hardware flow control configuration may be determined with the following *m336ctl* command.

m336ctl status /dev/tty*XY*

In this example, if hardware flow control is enabled for the specified port, the following message will be sent to standard output:

```
m336ctl: hardware handshake is: on
```

If hardware flow control is disabled for the specified port, the following message will be sent to standard output:

```
m336ctl: hardware handshake is: off
```

FILES

/dev/ttya*
/dev/ttyb*
/dev/ttyc*
/dev/ttyd*
/dev/ttye*
/dev/ttyf*

SEE ALSO

termios(7), tty(7), mvme336(7)
stty(1) in the *User's Reference Manual.*
ioctl(2) in the *Programmer's Reference Manual.*

NAME

m350ctl – MVME350 control program

SYNOPSIS

m350ctl [**–retwgbB**] [**–f**x] [**–s** [n [**kb**]]] [special]

DESCRIPTION

M350ctl controls function of the MVME350 streaming tape device. The following options are interpreted by *m350ctl:*

–r
Rewind tape.

–e
Erase tape.

–t
Retension tape.

–w
Open tape for writing (with filemark).

–g
Print DMA buffer size.

–b
Swap byte order for tape read/write.

–B
Set byte order for tape read/write to default (no swap).

–f*x*
Position tape at the front of file *x*.

–s*n*
Set DMA buffer size. The buffer size is set to *n* bytes, *n***b** (or *n***B**) blocks, or *n***k** (or *n*K) Kbytes. If *n* is not specified, the buffer size set to default value of 128 Kbytes. If *n* is zero, then double buffering is turned off. It is not recommended to change the buffer size unless the system contains less than 512 Kb, in which case the size should be reduced.

If the special file is not given, standard input will be used. For example, the command

```
m350ctl –e /dev/rmt/m350_0a
```

is identical to

m350ctl –e < /dev/rmt/m350_0a

If the **–w** option is used, the default special file is standard output. Thus, the command

m350ctl –ew /dev/rmt/m350_0t

is identical to

m350ctl –ew > /dev/rmt/m350_0t

The tape retension command

m350ctl –t /dev/rmt/m350_0a &

will run in the background.

To position the tape at the beginning of the first file on tape

m350ctl –f1 /dev/rmt/m350_0an

which will not rewind on close.

To find the beginning of the third file on a tape, use the command

m350ctl –f3 /dev/rmt/m350_0an

which will not rewind on close.

The command

m350ctl –f3 /dev/rmt/m350_0a

will also find the beginning of the third file on tape, but will rewind on close since the **n** option wasn't used.

See *mvme350*(7) for more information about special file naming conventions.

FILES

/dev/rmt/m350_*

SEE ALSO

mvme350(7)

NAME

m355ctl – MVME355 control program

SYNOPSIS

m355ctl [**–re**] [**–f***x*] [*special*]

DESCRIPTION

m355ctl controls the function of the MVME355 9-track tape device. The following options are interpreted by *m355ctl:*

–r
Rewind tape.

–e
Erase tape.

–f*x*
Position tape at the front of file *x*.

If the special file is not given, standard input will be used. For example, the command:

m355ctl –e /dev/rmt/m355_0 is identical to
m355ctl –e < /dev/rmt/m355_0

To position the tape at the beginning of the first file on tape, type the following, which will not rewind on close:

m355ctl –f1 /dev/rmt/m355_0n

To find the beginning of the third file on a tape, use the following command, which will not rewind on close:

m355ctl –f3 /dev/rmt/m355_0n

The following command also finds the beginning of the third file on tape, but will rewind on close because the **n** option was not used:
m355ctl –f3 /dev/rmt/m355_0

See *mvme355*(7) for more information about special file naming conventions.

FILES

/dev/rmt/m355_*

SEE ALSO

mvme355(7)

DIAGNOSTICS

m355ctl will complain, `too many requests` if the user requests more than one type of tape activity on the command line.

m355ctl will complain, `no request found` if the user does not request any type of tape activity on the command line.

m355ctl will complain, `too many arguments` if unexpected command line arguments are encountered.

If other errors are encountered in attempting to access the 9-track tape drive, *m355ctl* prints error messages via the standard *perror*() library routine.

WARNINGS

If *special* is given as a **/dev** entry for a non-MVME355 tape device, the behavior is unpredictable. The user is strongly cautioned not to use *m355ctl* with non-MVME355 devices.

NAME

makefsys – create a file system on a diskette

SYNOPSIS

makefsys

DESCRIPTION

This command allows the user to create a file system on a diskette. It also writes an internal label in the file system super-block.

The user is asked some questions before the file system is created. Once created, the diskette is self-identifying.

The identical function is available under the *sysadm* menu:

sysadm makefsys

The command may be assigned a password. See *sysadm*(1), the **admpasswd** sub-command.

SEE ALSO

checkfsys(1M), labelit(1M), mkfs(1M), mountfsys(1M), sysadm(1)

NAME

mkcomply – force compliance with a bill of materials

SYNOPSIS

/local/bin/mkcomply [-bclv] [*file*]

DESCRIPTION

mkcomply, without the -b or -c option, lists the differences between the entry as read from the bill of materials and the actual corresponding file. Any differences between the owner, group, mode, size, and check sum are reported. Any differences in link counts are not reported unless the -l option is used. *mkcomply* checks directories and special files in a similar way, except that sizes and check sums are not compared. For special files the device field in the bill of materials is checked against the actual file.

With the -c option, *mkcomply* changes the owner, group, and mode of the file as read from the bill of materials file. It does not check for any differences between the bill of materials file and the actual file when using this option.

If no file is given on the command line, *mkcomply* reads standard input.

The command line options are:

-b
: Print a bill of materials list. *mkcomply* expects to read a list of file names.

-c
: Force compliance on owner, group, and mode. *mkcomply* expects to read a bill of materials list.

-l
: Report any differences between the link counts in the bill of materials file and the corresponding actual files. This option is effective only when not using the -b or -c options.

-v
: Verbose. Print a list of the files being processed. This option is effective only when a bill of materials is being read.

DIAGNOSTICS

Exit status is 0 if no differences or errors are reported. Otherwise, exit status is the count of the number of errors recorded.

EXAMPLES

The following lines, excluding the heading line, are a sample of the input for *mkcomply*:

owner	group	mode	device	links	size	check sum	file name
0	3	060644	0700	3	0	0000	/dev/device1
0	3	020644	1a00	3	0	0000	/dev/device2
0	3	020644	1a02	3	0	0000	/dev/device3
0	3	060644	0702	3	0	0000	/dev/usr
0	3	040700	0000	2	0	0000	directory
3	3	100744	0000	2	23	6725	apple
2	2	100777	0000	2	8307	2efc	plum
3	3	100666	0000	1	771	3d35	orange

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs special blocks[:i-nodes] [gap blocks/cyl]
/etc/mkfs special proto [gap blocks/cyl]
/etc/mkfs[1248]k special blocks[:i-nodes] [gap blocks/cyl]
/etc/mkfs[1248]k special proto [gap blocks/cyl]

DESCRIPTION

mkfs constructs a file system by writing on the *special* file using the values found in the remaining arguments of the command line. The command waits 10 seconds before starting to construct the file system. During this 10 second pause you can abort the command by entering your defined interrupt key (`DEL`, `CTL+C`).

If the second argument is a string of digits, the file system size is the value of *blocks* interpreted as a decimal number. This is the number of *physical* (512 byte) disk blocks the file system will occupy. If the number of i-nodes is not given, the default varies with the logical block size of the file system. (The size of a *logical* block is determined by command name: **mkfs** and **mkfs1k=1024b; mkfs2k=2048b; mkfs4k=4096; mkfs8k= 8192b.**) The default number of inodes is computed by dividing the number of *logical* blocks by 4, 2, 1, and 1 for **mkfs** (and **mkfs1k**), **mkfs2k**, **mkfs4k**, and **mkfs8k**, repectively. *mkfs* builds a file system with a single empty directory on it. The boot program block (block 0) is left uninitialized.

If the second argument is the name of a file that can be opened, *mkfs* assumes it to be a prototype file *proto*, and takes its directions from that file. The prototype file contains tokens separated by spaces or new-lines. A sample prototype specification follows (line numbers have been added to aid in the explanation):

```
1.      /dev/null
2.      4872 110
3.      d--777 3 1
4.      usr    d--777 3 1
5.             sh     ---755 3 1 /bin/sh
6.             ken    d--755 6 1
7.                    $
8.             b0     b--644 3 1 0 0
9.             c0     c--644 3 1 0 0
10.            $
11.     $
```

Line 1 in the example is the name of a file to be copied onto block zero (on some systems, this may be used as the bootstrap program; it must be ≤ 512 bytes).

Line 2 specifies the number of *physical* (512 byte) blocks the file system is to occupy and the number of i-nodes in the file system.

Lines 3-9 tell *mkfs* about files and directories to be included in this file system.

Line 3 specifies the root directory.

Lines 4-6 and 8-9 specifies other directories and files.

The $ on line 7 tells *mkfs* to end the branch of the file system it is on, and continue from the next higher directory. The $ on lines 10 and 11 end the process, since no additional specifications follow.

File specifications give the mode, the user ID, the group ID, and the initial contents of the file. Valid syntax for the contents field depends on the first character of the mode.

The mode for a file is specified by a 6-character string. The first character specifies the type of the file. The character range is **–bcd** to specify regular, block special, character special and directory files respectively. The second character of the mode is either **u** or – to specify set-user-id mode or not. The third is **g** or – for the set-group-id mode. The rest of the mode is a 3 digit octal number giving the owner, group, and other read, write, execute permissions (see *chmod*(1)).

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

If the file is a regular file, the next token of the specification may be a path name whence the contents and size are copied. If the file is a block or character special file, two decimal numbers follow which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries for **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. As noted above, the scan is terminated with the token **$**.

The final argument in both forms of the command specifies the rotational *gap* and the number of *blocks/cyl*. The recommended values are listed in Appendix A in the *System Administrator's Guide*.

If the *gap* and *blocks/cyl* are not specified or are considered illegal values, a default value of gap size 1 and 400 blocks/cyl is used.

SEE ALSO

chmod(1) in the *User's Reference Manual*.
dir(4), fs(4) in the *Programmer's Reference Manual*.

BUGS

With a prototype file, it is not possible to copy in a file larger than 500K bytes, nor is there a way to specify links. The maximum number of i-nodes configurable is 65500.

NAME

mknod – build special file

SYNOPSIS

/etc/mknod *name* **b | c major minor**
/etc/mknod *name* **p**

DESCRIPTION

mknod makes a directory entry and corresponding i-node for a special file.

The first argument is the *name* of the entry. The convention is to keep such files in the **/dev** directory.

In the first case, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g., unit, drive, or line number). They may be either decimal or octal. The assignment of major device numbers is specific to each system. The information is contained in the system source file **conf.c**. You must be the super-user to use this form of the command.

The **-p** is the form of the *mknod* that is used to create FIFO's (a.k.a named pipes).

WARNING

If **mknod** is used to create a device in a remote directory (Remote File Sharing), the major and minor device numbers are interpreted by the server.

SEE ALSO

mknod(2) in the *Programmer's Reference Manual*.

NAME

mount, umount – mount and unmount file systems and remote resources

SYNOPSIS

/etc/mount [[-r] [-f *fstyp*] *special directory*]
/etc/mount [[-r] -f NFS[*,options*] *special directory*]
/etc/mount [[-r] [-c] -d *resource directory*]
/etc/umount *directory*
/etc/umount -d *resource*

DESCRIPTION

File systems other than root (/) are considered *removable* in the sense that they can be either available to users or unavailable. *mount* announces to the system that *special*, a block special device or *resource*, a RFS resource, is available to users from the mount point directory. *directory* must exist already; it becomes the name of the root of the newly mounted *special*.

mount, when entered with arguments, adds an entry to the table of mounted devices, **/etc/mnttab**. *umount* removes the entry. If invoked with no arguments, *mount* prints the entire mount table. If invoked with an incomplete argument list, *mount* searches **/etc/fstab** for the missing arguments.

The following options are available:

–r
indicates that *special* or *resource* is to be mounted read-only. If *special* or *resource* is write-protected, this flag must be used.

–c
indicates that remote reads and writes should not be cached in the local buffer pool. **-c** is used in conjunction with **-d**. By default, remote reads and writes will be cached if the version of RFS on the remote system supports client caching.

–d
indicates that *resource* is a RFS resource that is to be mounted on *directory* or unmounted. To mount a remote resource, RFS must be up and running and the resource must be advertised by a remote computer (see *rfstart*(1M) and *adv*(1M)).

–f

fstyp indicates that *fstype* is the file system type to be mounted. If this argument is omitted, it defaults to the **root** *fstyp*. If *fstyp* is NFS, then NFS options may be added after the *fstyp* separated by commas. The available NFS options are:

soft

return an error if the server doesn't respond. If a file system is mounted read-only, this option allows the client system to keep running by using only local data when a server system goes down. However, if a process is accessing remotely mounted data when the server goes down, using this option may result in loss of data. It is not recommended unless used with the **r** option. (The default is **hard**, which instructs the software to retry the request until the server responds.)

rsize=*n*

set the read buffer size to *n* bytes.

wsize=*n*

set the write buffer size to *n* bytes.

timeo=*n*

set the initial NFS timeout to *n* tenths of a second.

retrans=*n*

set the number of NFS retransmissions to *n*.

port=*n*

set the server IP port number to *n*.

special

indicates the block special device that is to be mounted on *directory*. If *fstyp* is NFS, then *special* should be of the form *hostname:/pathname*.

resource

indicates the remote resource name that is to be mounted on a directory.

directory

indicates the directory mount point for *special* or *resource*. (The directory must already exist.)

Umount announces to the system that the file system previously mounted *special* or *resource* is to be made unavailable. If invoked with an incomplete argument list, *umount* searches **/etc/fstab** for the missing arguments.

mount can be used by any user to list mounted file systems and resources. Only a superuser can mount and umount file systems.

FILES

/etc/mnttab	mount table
/etc/fstab	file system table

SEE ALSO

adv(1M), fuser(1M), rfstart(1M), setmnt(1M), unadv(1M) in the *System Administrator's Reference Manual*.
mountd(1M), nfsd(1M), showmount(1M) in the *NSE System Administrator's Reference Manual*.
mount(2), umount(2), fstab(4), mnttab(4N) in the *Programmer's Reference Manual*.
Remote File Sharing chapter, *System Administrator's Guide* for guidelines when mounting remote resources.

WARNINGS

Physically removing a mounted file system diskette from the diskette drive before issuing the *umount* command damages the file system.

DIAGNOSTICS

If the *mount(2)* system call fails, *mount* prints an appropriate diagnostic. *mount* issues a warning if the file system to be mounted is currently mounted under another name. A remote resource mount will fail if the resource is not available or if RFS is not running.

umount fails if *special* or *resource* is not mounted or if it is busy. *special* or *resource* is busy if it contains an open file or some user's working directory. In such a case, you can use *fuser*(1M) to list and kill processes that are using *special* or *resource*.

NAME

mountall, umountall – mount, unmount multiple file systems

SYNOPSIS

/etc/mountall [–] [file-system-table] ...
/etc/umountall [–k]

DESCRIPTION

These commands may be executed only by the superuser.

mountall is used to mount file systems according to a *file-system-table*. (**/etc/fstab** is the default file system table.) The special file name "–" reads from the standard input.

Before each file system is mounted, it is checked using *fsstat*(1M) to see if it appears mountable. If the file system does not appear mountable, it is checked, using *fsck*(1 M), before the mount is attempted.

umountall causes all mounted file systems except **root** to be unmounted. The **–k** option sends a SIGKILL signal, via *fuser*(1M), to processes that have files open.

FILES

File-system-table format:

column 1	block special file name of file system
column 2	mount-point directory
column 3	"–r" if to be mounted read-only; "–d" if remote
column 4	(optional) file system type string
column 5+	ignored

White-space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

A typical file-system-table might read:

```
/dev/dsk/c1d0s2    /usr -r S51K
```

SEE ALSO

fsck(1M), fsstat(1M), fuser(1M), mount(1M)
sysadm(1) in the *User's Reference Manual.*
signal(2), fstab(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

No messages are printed if the file systems are mountable and clean.

Error and warning messages come from *fsck*(1M), *fsstat*(1M), and *mount*(1M).

NAME

mountfsys, umountfsys – mount, unmount a diskette file system

SYNOPSIS

mountfsys [**–y**] [**–r**]
umountfsys [**–y**]

DESCRIPTION

The *mountfsys* command mounts a file system that is on a removable disk so that users can read and write on it. The options provide the following:

–r
the file system is mounted read-only.

–y
suppresses any questions asked during mounting or unmounting.

The *umountfsys* command unmounts the file system.

By default, the name of the file system is displayed and the user is asked if it should be mounted. The optional **–y** argument suppresses questions and mounts or unmounts the file system immediately.

The identical functions are available under the *sysadm* menu:

sysadm mountfsys
sysadm umountfsys

The commands may be assigned passwords. See *sysadm*(1), the **admpasswd** sub-command.

SEE ALSO

checkfsys(1M), mount(1M), makefsys(1M)
sysadm(1) in the *User's Reference Manual.*

WARNING

ONCE THE DISK IS MOUNTED IT MUST NOT BE REMOVED FROM THE DISK DRIVE UNTIL IT HAS BEEN UNMOUNTED!

Removing the disk while it is still mounted can cause severe damage to the data on the disk.

BUGS

A file system that has no label cannot be mounted with the *mountfsys* command.

NAME

mvdir – move a directory

SYNOPSIS

/etc/mvdir *dirname* *name*

DESCRIPTION

mvdir moves directories within a file system. *Dirname* must be a directory. If *name* does not exist, it will be created as a directory. If *name* does exist, and is a directory, *dirname* will be created as *name/dirname*. *Dirname* and *name* may not be on the same path; that is, one may not be subordinate to the other. For example:

```
mvdir x/y x/z
```

is legal, but

```
mvdir x/y x/y/z
```

is not.

SEE ALSO

mkdir(1), mv(1) in the *User's Reference Manual*.

WARNINGS

Only the superuser can use *mvdir*.

NAME

ncheck – generate path names from i-numbers

SYNOPSIS

/etc/ncheck [**–i** *i-numbers*] [**–a**] [**–s**] [*file-system*]

DESCRIPTION

ncheck with no arguments generates a path-name vs. i-number list of all files on a set of default file systems (see */etc/checklist*). Names of directory files are followed by /..

The options are:

–i
limits the report to only those files whose i-numbers follow.

–a
allows printing of the names . and .., which are ordinarily suppressed.

–s
limits the report to special files and files with set-user-ID mode. This option may be used to detect violations of security policy.

File system must be specified by the file system's special file.

The report should be sorted so that it is more useful. For example:

```
ncheck file_system | sort +1 –f –onchecklist
```

places the output of **ncheck** in a file called *nchecklist* in alphabetic order, ignoring case during the **sort**(1) function. The command:

```
ncheck file_system | sort –n –onchecklist
```

can be used to produce a list sorted by i-node number.

SEE ALSO

fsck(1M), fsdb(1M)
sort(1) in the *User's Reference Manual.*

DIAGNOSTICS

If the file system structure is not consistent, ?? denotes the "parent" of a parentless file and a path-name beginning with ... denotes a loop.

NAME

newgrp – log in to a new group

SYNOPSIS

newgrp [–] [group]

DESCRIPTION

newgrp changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp*, regardless of whether it terminated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (PS1) other than **$** (default) and has not exported PS1. After an invocation of *newgrp* , successful or not, their PS1 will now be set to the default prompt string **$**. Note that the shell command *export* (see *sh*(1)) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry. This is a way to exit the effect of an earlier *newgrp* command.

If the first argument to *newgrp* is a –, the environment is changed to what would be expected if the user actually logged in again as a member of the new group.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group. The operating system uses POSIX multiple groups whereby a user simultaneously maintains membership in all the groups the user is a member of. However, since the group id of files created in a multiple group environment is set to the current effective group id, *newgrp* provides a mechanism to change this value, thus changing the group id of newly created files.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), sh(1) in the *User's Reference Manual*.
group(4), passwd(4), environ(5) in the *Programmer's Reference Manual*.

BUGS

There is no convenient way to enter a password into **/etc/group**. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

nlsadmin – network listener service administration

SYNOPSIS

nlsadmin –x

nlsadmin [**-q**] [**-v**] [**-z**] [**-l**] [**-t**] [**-i**] [**-m**] [**-p**] [**-w**] [**-c**] [**-y**] [**r**] [**-e**] [**-d**] [**-s**] [**-k**] *net_spec*

DESCRIPTION

nlsadmin administers the network listener process(es) on a machine. Each network has a separate instance of the network listener process associated with it; each instance (and thus, each network) is configured separately. The listener process "listens" to the network for service requests, accepts requests when they arrive, and spawns servers in response to those service requests. The network listener process will work with any network (more precisely, with any transport provider) that conforms to the transport provider specification.

The listener supports two classes of service: a general listener service, serving processes on remote machines, and a terminal login service, for terminals connected directly to a network. The terminal login service provides networked access to this machine in a form suitable for terminals connected directly to the network. However, this direct terminal service requires special associated software, and is only available with some networks (for example, the AT&T STARLAN network).

nlsadmin can establish a listener process for a given network, configure the specific attributes of that listener, and start and kill the listener process for that network. *nlsadmin* can also report on the listener processes on a machine, either individually (per network) or collectively.

The following list shows how to use *nlsadmin*. In this list, *net_spec* represents a particular listener process. Specifically, *net_spec* is the relative path name of the entry under **/dev** for a given network (that is, a transport provider). Changing the list of services provided by the listener produces immediate changes, while changing an address on which the listener listens has no effect until the listener is restarted. The following combination of options can be used.

nlsadmin
gives a brief usage message.

nlsadmin –x
reports the status of all of the listener processes installed on this machine.

nlsadmin *net_spec*
prints the status of the listener process for *net_spec.*

nlsadmin –q *net_spec*
queries the status of the listener process for the specified network, and will reflect the result of that query in its exit code. If a listener process is active, *nlsadmin* will exit with a status of 0; if no process is active, the exit code will be 1; the exit code will be greater than 1 in case of error.

nlsadmin –v *net_spec*
prints a verbose report on the servers associated with *net_spec*, giving the service code, status, command, and comment for each. It also specifies the **uid** the server will run as, and the list of modules to be pushed, if any, before the server is started.

nlsadmin –z *service_code net_spec*
prints a report on the server associated with *net_spec* that has service code *service_code*, giving the same information as in the **–v** option.

nlsadmin –q –z *service_code net_spec*
queries the status of the service with service code *service_code* on network *net_spec*, and will exit with a status of 0 if that service is enabled, 1 if that service is disabled, and greater than 1 in case of error.

nlsadmin –l *addr net_spec*
changes or set the address on which the listener listens (the general listener service). This is the address generally used by remote processes to access the servers available through this listener (see the **–a** option, below). *addr* is the transport address on which to listen and is interpreted using a syntax that allows for a variety of address formats. By default *addr* is interpreted as the symbolic ASCII representation of the transport address. An *addr* preceded by a **\x** will let you enter an address in hexadecimal notation. Note that *addr* must appear as a single word to the shell and must be quoted if it contains any blanks.

If *addr* is just a dash ("–"), *nlsadmin* reports the address currently configured, instead of changing it.

A change of address will not take effect until the next time the listener for that network is started.

nlsadmin –t *addr net_spec*
changes or sets the address on which the listener listens for requests for terminal service, but is otherwise similar to the **–l** option above. A terminal service address should not be defined unless the appropriate remote login software is available; if such software is available, it must be configured as service code 1 (see the **–a** option, below).

nlsadmin –i *net_spec*
initializes or change a listener process for the network specified by *net_spec*, that is, it will create and initialize the files required by the listener. Note that the listener should only be initialized once for a given network, and that doing so does not actually invoke the listener for that network. The listener must be initialized before assigning addressing or services.

nlsadmin [**–m**] **–a** *service_code [*–**p** *modules] [*–**w** *id*] **–c** *cmd* **–y** *comment net_spec*
adds a new service to the list of services available through the indicated listener. *service_code* is the code for the service, *cmd* is the command to be invoked in response to that service code, comprised of the full path name of the server and its arguments, and *comment* is a brief (free-form) description of the service for use in various reports. Note that *cmd* must appear as a single word to the shell, so if arguments are required the *cmd* and its arguments must be surrounded by quotes. Similarly, the *comment* must also appear as a single word to the shell. When a service is added, it is initially enabled (see the **–e** and **–d** options, below).

If the **–m** option is specified, the entry will be marked as an administrative entry. Service codes 1 through 100 are reserved for administrative entries, which are those that require special handling internally. In particular, code 1 is assigned to the remote login service, which is the service automatically invoked for connections to the terminal login address.

The **–m** option used with the **–a** option indicates that special handling internally is required for those servers added with the **–m** set. This internal handling is in the form of code embedded on the listener process.

If the –p option is specified, then *modules* will be interpreted as a list of STREAMS modules for the listener to push before starting the service being added. The modules are pushed in the order they are specified. *modules* should be a comma-separated list of modules, with no white space included.

If the **–w** option is specified, then *id* is interpreted as the user name from **/etc/passwd** that the listener should look up. From the user name, the listener should obtain the user ID, the group ID, and the home directory for use by the server. If **–w** is not specified, the default is to use the user ID **listen**.

A service must explicitly be added to the listener for each network on which that service is to be available. This operation will normally be performed only when the service is installed on a machine, or when populating the list of services for a new network.

nlsadmin –r *service_code net_spec*
removes the entry for the *service_code* from that listener's list of services. This will normally be performed only in conjunction with the de-installation of a service from a machine.

nlsadmin –e *service_code net_spec*
nlsadmin –d *service_code net_spec*
enables or disables (respectively) the service indicated by *service_code* for the specified network. The service must have previously been added to the listener for that network (see the **–a** option, above). Disabling a service will cause subsequent service requests for that service to be denied, but the processes from any prior service requests that are still running will continue unaffected.

nlsadmin –s *net_spec*
nlsadmin –k *net_spec*
starts and kills (respectively) the listener process for the indicated network. These operations will normally be performed as part of the system startup and shutdown procedures. Before a listener can be started for a particular network, it must first have been initialized, and an address must be defined for the general listener service (see the **–i** and **–l** options, above). When a listener is killed, processes that are still running as a result of prior service requests will continue unaffected.

The listener runs as user ID **root**, with group ID **sys**. A special ID, user ID **listen** and group ID **adm**, should be entered in the **/etc/passwd** file as a default ID for servers. The listener always uses as its home directory **/usr/net/nls**, which is concatenated with *net_spec* to determine the location of the listener configuration information for each network. The home directory specified in the **/etc/passwd** entry for **listener** will be used by servers that run as ID **listen**.

nlsadmin may be invoked by any user to generate reports, but all operations that affect a listener's status or configuration are restricted to the superuser.

FILES

/usr/net/nls/*net_spec*

SEE ALSO

Network Programmer's Guide, Vol. 1.

NAME

nsquery – Remote File Sharing name server query

SYNOPSIS

nsquery [-h] [*name*]

DESCRIPTION

nsquery provides information about resources available to the host from both the local domain and from other domains. All resources are reported, regardless of whether the host is authorized to access them. When used with no options, *nsquery* identifies all resources in the domain that have been advertised as sharable. A report on selected resources can be obtained by specifying *name*, where *name* is:

nodename
The report will include only those resources available from *nodename*.

domain.
The report will include only those resources available from *domain*.

domain.nodename
The report will include only those resources available from *domain.nodename*.

When the name does not include the delimiter ".", it will be interpreted as a *nodename* within the local domain. If the name ends with a delimiter ".", it will be interpreted as a domain name.

The information contained in the report on each resource includes its advertised name (domain.resource), the read/write permissions, the server (nodename.domain) that advertised the resource, and a brief textual description.

When *–h* is used, the header is not printed.

A remote domain must be listed in your **rfmaster** file in order to query that domain.

EXIT STATUS

If no entries are found when *nsquery* is executed, the report header is printed.

ERRORS

If your host cannot contact the domain name server, an error message will be sent to standard error.

SEE ALSO

adv(1M), unadv(1M)
rfmaster(4) in the *Programmer's Reference Manual*.

NAME

passmgmt – password files management

SYNOPSIS

passmgmt –a *options name*
passmgmt –m *options name*
passmgmt –d -c -h -u -o -g -s -l *name*

DESCRIPTION

The *passmgmt* command updates information in the password files. This command works with both **/etc/passwd** and **/etc/shadow**. If there is no **/etc/shadow**, the changes done by *passmgmt* will only go to **/etc/passwd**.

passmgmt –a adds an entry for user *name* to the password files. This command does not create any directory for the new user and the new login remains locked (with the string ***LK*** in the password field) until the *passwd*(1) command is executed to set the password.

passmgmt –m modifies the entry for user *name* in the password files. The name field in the **/etc/shadow** entry and all the fields (except the password field) in the **/etc/passwd** entry can be modified by this command. Only fields entered on the command line will be modified.

passmgmt –d deletes the entry for user *name* from the password files. It will not remove any files that the user owns on the system; they must be removed manually.

The following options are available:

–c *comment*
A short description of the login. It is limited to a maximum of 128 characters and defaults to an empty field.

–h *homedir*
Home directory of *name*. It is limited to a maximum of 256 characters and defaults to **/usr/***name*.

–u *uid*
UID of the *name*. This number must range from 0 to the maximum non-negative value for the system. It defaults to the next available UID greater than 99. Without the **–o** option, it enforces the uniqueness of a UID.

–o
This option allows a UID to be non-unique. It is used only with the –u option.

–g *gid*
GID of the *name*. This number must range from 0 to the maximum non-negative value for the system. The default is 1.

–s *shell*
Login shell for *name*. It should be the full pathname of the program that will be executed when the user logs in. The maximum size of *shell* is 256 characters. The default is for this field to be empty and to be interpreted as **/bin/sh**.

–l *logname*
This option changes the *name* to *logname*. It is used only with the –m option.

The total size of each login entry is limited to a maximum of 511 bytes in each of the password files.

FILES

/etc/passwd, /etc/shadow, /etc/opasswd, /etc/oshadow

SEE ALSO

passwd(4) in the *System Aministrator's Reference Manual*.
passwd(1) in the *User's Reference Manual*.

DIAGNOSTICS

The *passmgmt* command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid command syntax. Usage message of the **passmgmt** command will be displayed.
3	Invalid argument provided to option.
4	UID in use.
5	Inconsistent password files (e.g., *name* is in the **/etc/passwd** file and not in the **/etc/shadow** file, or vice versa).
6	Unexpected failure. Password files unchanged.
7	Unexpected failure. Password file(s) missing.
8	Password file(s) busy. Try again later.
9	*name* does not exist (if **–m** or **–d** is specified), already exists (if **–a** is specified), or *logname* already exists (if **–m –l** is specified).

NOTE

You cannot use a colon or <CR> as part of an argument because it will be interpreted as a field separator in the password file.

NAME

patch – examine and change bytes in a file

SYNOPSIS

bpatch [<*filename*>]

DESCRIPTION

bpatch allows a user to look at specific areas of a file and to change selected bytes. *bpatch* takes no information from the command line except for the optional name of a file to open initially. *bpatch* displays a prompt > and waits for user input. The following commands are recognized:

o filename
Open the specified file. If a file is already open, it is closed.

s xxxx yyyy
Display the contents of the file from **xxxx** to **yyyy** , where **xxxx** and **yyyy** are hex values. The starting address is rounded down modulo hex 10; the ending address is rounded up modulo hex 10. The contents are shown both as hex and printed ASCII. Non-printing characters are shown as periods in the ASCII display.

c xxxx zz ...
Change location **xxxx** to **zz** , where **xxxx** and **zz** are hex numbers. Several values can be changed by entering additional values. Each additional value changes the next sequential byte.

d
Prints information about the file e.g., type, permissions, size.

q
Exit.

The hex numbers, **xxxx** and **yyyy** , can represent a full 32 bit address, i.e., eight hex characters. The hex number, **zz** , should be only two hex characters because it represents a byte.

Bpatch is insensitive to case of alphabetic characters. The above commands, as well as the characters **[A–F]** within hex numbers, may be upper- or lowercase.

FILES

None.

EXAMPLES

```
s 0 ff

S 10000 100FF

c 400 7F 31 21

o newfile

d

q
```

DIAGNOSTICS

bpatch gives error messages if it is unable to open a file, or if any commands are entered that would read or write when no file is open.

If an illegal command, or no input is entered, *bpatch* displays a brief help message.

If the user has read permission, but not write permission, on a file, *bpatch* displays a message to this effect at open time. The user can look at the file, but may not use the **c** command.

NAME

port_hold - hold serial lines open

SYNOPSIS

/local/bin/port_hold *<stty_file >log_file 2>&1 &*

DESCRIPTION

port_hold, which runs as a background process started by **/etc/rc2.d/S80***port_hold,* opens and establishes initial *stty*(1) modes for each port named in *stty_file.* Signals are trapped and ignored, so *port_hold* never terminates.

port_hold can be used to compensate for a "feature" of the serial port subsystem. As soon as no process has a particular serial port open, the line discipline for that port immediately reverts to a default state (which does not honor XON/XOFF flow control), even if data is still draining to the port. If the serial device attempts to throttle the system's output by transmitting an XOFF at a time when no process has the port open, the XOFF is ignored; output continues; and the device is usually overrun. This behavior tends to cause problems for serial printers: the symptom is missing and/or garbled printer output, especially near the end of an otherwise correct print job. This symptom may be observed only rarely, due to the narrow timing window in which the "problem" occurs.

It should be noted that however troublesome this is, it exists for good, although arcane, reasons. Likewise, the default line discipline for a closed port is proper, if somewhat inconvenient. *port_hold* overcomes the problems inherent in this feature by making sure that at least one process holds the port open at all times, with the correct line discipline. *port_hold* accomplishes this at the lowest possible cost: only one process is required to hold any number of ports open perpetually; each port can have a different "default" *stty* setting, as required.

stty_file is an ASCII text file, each line of which is either a comment or a *stty*(1) command line. Comment lines, which must start with a pound sign (#), are ignored. All other lines are expected to be valid *stty* commands, which must start with the string *stty* and end with a standard input redirection clause (i.e., **</dev/ttyxx**). The device named in the redirection must be a character special device. Lines which do not meet these requirements are ignored.

The following processing is done for each valid *stty* line in *stty_file:*

(1) Open the port specified by the input redirection clause. NDELAY mode is used so that the open will not wait for carrier (DCD).

(2) Set **clocal** mode on the port so that further processing will not require carrier (DCD).

(3) Fork the *stty* command to establish initial modes for the port.

When all the above processing is completed, *port_hold* executes a *pause*(2) system call. Since it never terminates, the ports opened by *port_hold* will never be closed.

Good things to include always in the *stty* line are: **ixon**, **ixoff**, **-ixany**, and **clocal.** Other items are appropriate if the device expects a 7-bit character size, parity checking, and so on.

NOTE

Because *port_hold* traps all signals, it must finish all of its processing by the time **/etc/rc2.d/S80***port_hold* terminates to avoid "interrupted system call" errors. For this reason, execution of *port_hold* is typically followed by a *sleep*(1) of sufficient duration to assure that *port_hold* enters the pause state before the sleep expires. Note also that the file system (usually **/usr/tmp**) which is to contain the log file must be mounted before *port_hold* is started.

EXAMPLES

A sample **/etc/rc2.d/S80port_hold** is listed below:

```
if [ -s /etc/psttys ] then          /local/bin/port_hold </etc/psttys
>/usr/tmp/port_hold.log 2>&1 &       sleep 5
```

where **/etc/psttys** might look like this:

```
# list of ports and stty lines for port_hold
stty ixon ixoff -ixany clocal </dev/tty01
stty ixon ixoff -ixany clocal cs7 evenp parenb </dev/tty02
```

port_hold is started only if the *stty_file,* in this case **/etc/psttys** exists and has a non-zero length. Since there is no reasonable set of defaults, no *stty_file* is provided with the operating system; this file must be constructed as needed.

Note that *port_hold* will start running as the system enters run level 2. There is no special provision for stopping *port_hold* at any other run level (nor is there any need to do so).

FILES

stty_file is best kept in **/etc**. Since it is a text file, it can be maintained with any available editor. It should be owned by root and have permissions set to octal 444.

log_file is best kept in **/usr/tmp**. See DIAGNOSTICS, below, for information on contents.

SEE ALSO

init(1M), termio(7), termios(7)
stty(1) in the *User's Reference Manual*.
intro(2) in the *Programmer's Reference Manual*.

DIAGNOSTICS

Messages concerning various problems that may occur in reading *stty_file*, opening ports, forking sttys, appear in *log_file*. Most messages are self-explanatory, but one deserves further clarification:

error 4 from fork or exec of *stty_command*

Indicates that **/etc/rc2.d/S80***port_hold* terminated before *port_hold* completed all of its processing (error code 4 indicates an "interrupted system call"). In such cases, the *sleep(1)* period should be increased.

NAME

portconfig, portdisplay - configure or display *tty* ports

SYNOPSIS

/etc/portconfig {-t | -m}
/etc/portdisplay

DESCRIPTION

/etc/portconfig creates character device files in **/dev** for all configured (hardware and software) RS-232 Serial Communications Boards on a system.

portconfig should be run in *single user mode* with **/usr** mounted.

The 332XT, and 335 ports are named:

tty[module#][port#]

Modules are numbered **1** through **9** and correspond to the relative (right-to-left) position of the rear panel transition module providing the DB-25 serial connections to a communications board. These *logical* module numbers *do not* correspond one to one with the *physical slot* numbers which are likely printed on the rear of the machine. They refer only to serial transition modules and are relative to the right-most one (numbered 1), regardless of the physical slot(s) it may occupy.

Ports are numbered **1** through *n*, where *n* is the number of ports supported by the corresponding communications board (and transition module).

The 336 ports are named:

tty[server#][port#]

Servers are named **a** through **f**. **Ports** are numbered **01** through **16**. These ports are made independent of the 332XT, and 335.

Please note that Transition modules are variously referred to as *Transition Boards, Transceiver Modules, Distribution Modules,* and *Distribution Cards.* This document uses the term **module**.

portconfig does the following:

- Creates TTY device files for each configured serial communications board.
- Removes any TTY device files for a board which no longer exists or for which no driver exists in the current kernel.

- Prints a pictorial representation of what the rear panel distribution module layout should be, based on the board set found.
- Creates the *portdisplay* command which prints, on demand, the current rear panel configuration (see examples below).

The **-t** option makes all changes to **/tmp/dev** and creates **/tmp/portdisplay** and **/tmp/inittab**.

The **-m** option causes updates to be made directly to **/dev** and creates the command **/etc/portdisplay** and modifies **/etc/inittab** (if an MVME336 controller is present and supported on the system).

The current hardware/software board configuration is determined as follows:

1. For each board type, determine if a driver for that board is configured into the most recently generated kernel. This is done by looking for a driver entry in **/usr/src/uts/mot/cf/m88k/master**. This file is created by *sysgen*(1M).
2. For each driver found above, determine if any associated boards are installed on (i.e., configured into) the system. This is done by creating a set of test nodes, one for each possible instance of the board type, and attempting to access (open) that device. A successful open implies the board is installed.

portconfig makes some important assumptions:

1. The system is booted on the kernel described by the above **master** file; otherwise, the hardware probes fail. *portconfig* does not currently distinguish between the absence of a software driver and the absence of the board.
2. The above **master** file accurately reflects the hardware complement (and rear panel configuration).

If there are boards and transition modules installed for which no driver is configured, the rear panel configuration assumed (and displayed) by *portconfig* **will not match the system.**

The following board types are currently configurable via *portconfig*:

Board	Transition Module	Number of Ports
MVME332XT	MVME710	8
MVME335	MVME715P	4
MVME336	MVME751	96

A sample pictorial output follows. In this example, the tty device nodes **tty2[1-4]** and **tty1[1-8]** would be created:

```
   715/335       1437/332xt
  __________     __________
 |   _      |   |  _    _  |
 |  | |     |   | | |  | | |
 |  | |  ___|   | | |  | | |
 |  |_| |  ||   | |_|  |_| |
 |  sp4 |p ||   | sp7  sp8 |
 |   _  |t ||   |  _    _  |
 |  | | |r ||   | | |  | | |
 |  | | |__||   | | |  | | |
 |  |_|     |   | |_|  |_| |
 |  sp3     |   | sp5  sp6 |
 |   _      |   |  _    _  |
 |  | |     |   | | |  | | |
 |  | |     |   | | |  | | |
 |  |_|     |   | |_|  |_| |
 |  sp2     |   | sp3  sp4 |
 |   _      |   |  _    _  |
 |  | |     |   | | |  | | |
 |  | |     |   | | |  | | |
 |  |_|     |   | |_|  |_| |
 |  sp1     |   | sp1  sp2 |
 |__________|   |__________|

  (module 2)     (module 1)     (these numbers are not displayed)
   tty2[1-4]      tty1[1-8]
```

The addition of another MVME332XT board (and SMM1437 transition module) would cause portconfig to assume the configuration:

```
   715/335           1437/332xt        1437/332xt

 ____________      ____________      ____________
|   _        |    |  _    _    |    |  _    _    |
|  | |       |    | | |  | |   |    | | |  | |   |
|  | |   ___ |    | | |  | |   |    | | |  | |   |
|  |_|  |   ||    | |_|  |_|   |    | |_|  |_|   |
|  sp4  |p  ||    | sp7  sp8   |    | sp7  sp8   |
|   _   |t  ||    |  _    _    |    |  _    _    |
|  | |  |r  ||    | | |  | |   |    | | |  | |   |
|  | |  |___||    | | |  | |   |    | | |  | |   |
|  |_|       |    | |_|  |_|   |    | |_|  |_|   |
|  sp3       |    | sp5  sp6   |    | sp5  sp6   |
|   _        |    |  _    _    |    |  _    _    |
|  | |       |    | | |  | |   |    | | |  | |   |
|  | |       |    | | |  | |   |    | | |  | |   |
|  |_|       |    | |_|  |_|   |    | |_|  |_|   |
|  sp2       |    | sp3  sp4   |    | sp3  sp4   |
|   _        |    |  _    _    |    |  _    _    |
|  | |       |    | | |  | |   |    | | |  | |   |
|  | |       |    | | |  | |   |    | | |  | |   |
|  |_|       |    | |_|  |_|   |    | |_|  |_|   |
|  sp1       |    | sp1  sp2   |    | sp1  sp2   |
|____________|    |____________|    |____________|

 (module 3)         (module 2)         (module 1)
 tty3[1234]         tty2[1-8]          tty1[1-8]
                      (new)
```

(these numbers are not disp

It would then be necessary to move any attached cables on the 715 (335) module to their corresponding location on the new 332XT module to its right. Thus all the cables retain their original **tty** names. The addition of this board/module would create eight new **tty** nodes for use; four served by the MVME332XT (**tty2[5678]**) and four served by the MVME335 (**tty3[1234]**).

The algorithm used by *portconfig* to assign **tty** names is predicated on the Rear Panel Card Placement rules outlined in the *System Manual* for your system. These rules are based on the Board Placement rules (also outlined in those documents) which dictate where boards should be placed on the backplane. The rear panel rules can be briefly summarized as:

> The smaller the number of ports provided by a transition module, the farther to the left (increasing relative module numbers) it resides. New modules are inserted to the left of existing, equivalent modules and to the right of existing, smaller modules. They bump the smaller modules to the left by one relative position and require any connected cables to be moved to the right, preserving their **tty[module#][port#]** name.

WARNINGS

If *portconfig* detects an MVME336 controller, it appends MVME336 entries to the end of the targeted **inittab** (**/tmp/inittab** with *portconfig -t* or **/etc/inittab** with *portconfig -m*). All such entries created by *portconfig* are turned "off". The administrator must manually turn on desired ports in the new **inittab** after running *portconfig*. If no MVME336 board is present, any MVME336 entries that may be in the targeted **inittab** are removed.

portconfig does not make modifications to **/etc/inittab** for any nodes added/deleted. Turning entries on/off and adding new entries must be performed manually, via **sysadm**(1M) or the available text editors.

Entries for parallel ports are not created.

If more than nine modules are configured, *portconfig* begins creating node names of the form **tty1xy**. System software has not been exhaustively tested for intelligent behavior with three-digit **TTY** names.

Links to existing **tty** node names are not detected or deleted when deleting nodes for non-configured devices. Attempts to access the linked name for a port which is no longer configured result in error (if driver has been removed, errno = ENODEV - No such device; if addressed board is not installed, errno = ENXIO - No such device or address).

NOTE

The MVME336 DeltaLink Server must be powered up and connected to the DeltaLink Hub for *portconfig* to properly detect the ports.

FILES

/tmp/dev/tty*	Test directory/nodes
/tmp/devXXXXX	Test directory (XXXXX = processid)
/tmp/portdisplay	Test version of /etc/portdisplay

/usr/adm/ports/*	Transition board pictorials

SEE ALSO

System Manual for your system.

NAME

powerdown – stop all processes and turn off the power

SYNOPSIS

powerdown [–y | –Y]

DESCRIPTION

This command brings the system to a state where nothing is running and then turns off the power.

By default, the user is asked questions that control how much warning the other users are given. The options are:

–y
: prevents the questions from being asked and just gives the warning messages. There is a 60 second pause between the warning messages.

–Y
: is the same as –y except it has no pause between messages. It is the fastest way to bring the system down.

The identical function is also available under the *sysadm* command:

sysadm powerdown

Password control can be instituted on this command. See *sysadm*(1), **admpasswd** sub-command.

EXAMPLES

```
some-long-running-command;  powerdown –y
```

The first command is run to completion and then the machine turns off. This is useful for, say, formatting a document to the printer at the end of a day.

FILES

/etc/shutdown - invoked by powerdown

SEE ALSO

shutdown(1M)
sysadm(1) in the *User's Reference Manual*.

NAME

profiler: prfld, prfstat, prfdc, prfsnap, prfpr – UNIX system profiler

SYNOPSIS

/etc/prfld [*system_namelist*]
/etc/prfstat on
/etc/prfstat off
/etc/prfdc *file* [*period* [*off_hour*]]
/etc/prfsnap *file*
/etc/prfpr *file* [*cutoff* [*system_namelist*]]

DESCRIPTION

prfld, *prfstat*, *prfdc*, *prfsnap*, and *prfpr* form a system of programs to facilitate an activity study of the operating system.

prfld is used to initialize the recording mechanism in the system. It generates a table containing the starting address of each system subroutine as extracted from *system_namelist*.

prfstat is used to enable or disable the sampling mechanism. Profiler overhead is less than 1% as calculated for 500 text addresses. *prfstat* will also reveal the number of text addresses being measured.

prfdc and *prfsnap* perform the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed. *prfdc* will store the counters into *file* every *period* seconds and will turn off at *off_hour* (valid values for *off_hour* are **0–24**). *prfsnap* collects data at the time of invocation only, appending the counter values to *file*.

prfpr formats the data collected by *prfdc* or *prfsnap*. Each text address is converted to the nearest text symbol (as found in *system_namelist*) and is printed if the percent activity for that range is greater than *cutoff*.

FILES

/dev/prf
interface to profile data and text addresses

/unix
default for system namelist file

NAME

prtconf – print system configuration

SYNOPSIS

/etc/prtconf

DESCRIPTION

The *prtconf* command prints the system configuration information which includes the memory and peripheral configuration.

As an option, the configuration information can be displayed every time the system is initialized to the multi-user state. This option is turned on by copying the file **/etc/prtconf** to the file **/etc/prtconfig**. The default is 'on'; the option can be turned off by removing **/etc/prtconf**.

NAME

pwck, grpck – password/group file checkers

SYNOPSIS

/etc/pwck [*file*]
/etc/grpck [*file*]

DESCRIPTION

pwck scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and the program-to-use-as-Shell exist. The default password file is **/etc/passwd**.

Grpck verifies all entries in the group file. 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**.

FILES

/etc/group
/etc/passwd

SEE ALSO

group(4), passwd(4) in the *Programmer's Reference Manual*.

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

NAME

pwconv – install and update **/etc/shadow** with information from **/etc/passwd**

SYNOPSIS

pwconv

DESCRIPTION

The *pwconv* command creates and updates **/etc/shadow** with information from **/etc/passwd**.

If the **/etc/shadow** file does not exist, *pwconv* creates **/etc/shadow** with information from **/etc/passwd**. The command populates **/etc/shadow** with the user's login name, password, and password aging information. If password aging information does not exist in **/etc/passwd** for a given user, none will be added to **/etc/shadow**. However, the "last changed" information is always updated.

If the **/etc/shadow** file does exist, the following tasks are performed:

Entries that are in the **/etc/passwd** file and not in the **/etc/shadow** file are added to the **/etc/shadow** file.

Entries that are in the **/etc/shadow** file and not in the **/etc/passwd** file are removed from **/etc/shadow**.

Password attributes (e.g., password and aging information) that exist in an **/etc/passwd** entry are moved to the corresponding entry in **/etc/shadow.**

The *pwconv* program is a privileged system command that cannot be executed by ordinary users.

FILES

/etc/passwd, /etc/shadow, /etc/opasswd, /etc/oshadow

SEE ALSO

passwd(1), passmgmt(1M), pwunconv(1M)

DIAGNOSTICS

The *pwconv* command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid command syntax.
3	Unexpected failure. Conversion not done.
4	Unexpected failure. Password file(s) missing.
5	Password file(s) busy. Try again later.

NAME

pwunconv – converts from a two- to a one-password file scheme.

SYNOPSIS

pwunconv

DESCRIPTION

pwunconv converts a UNIX system from a two password file scheme (**/etc/passwd** and **/etc/shadow**) to a one password file scheme (**/etc/passwd**). It updates **/etc/passwd** with password information from **/etc/shadow.** If aging information is present in **/etc/shadow,** the password aging information in **/etc/passwd** is also updated.

The total size of a login entry for the password file is limited to a maximum of 511 bytes.

FILES

/etc/passwd, /etc/shadow, /etc/opasswd, /etc/oshadow

SEE ALSO

passwd(1M), passmgmt(1M), pwconv(1M)

DIAGNOSTICS

The *pwunconv* command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid command syntax.
3	Unexpected failure. Conversion not done.
4	Unexpected failure. Password file missing.
5	Password file(s) busy. Try again later.
6	Shadow password file does not exist.
7	Entry for *name* too long. Conversion not done.

NAME

rc0 – run commands performed to stop the operating system

SYNOPSIS

/etc/rc0

DESCRIPTION

This file is executed at each system state change that needs to have the system in an inactive state. It is responsible for those actions that bring the system to a quiescent state, traditionally called "shutdown".

There are three system states that require this procedure. They are state 0 (the system halt state), state 5 (the system is secured for powerdown/reset), and state 6 (essentially the same as state 5). Whenever there is a change to one of these states, the **/etc/rc0** procedure is run. The entry in **/etc/inittab** might read:

```
s0:056:wait:/etc/rc0 >/dev/console 2>&1 </dev/console
```

Some of the actions performed by **/etc/rc0** are carried out by files in the directory **/etc/shutdown.d** and files beginning with K in **/etc/rc0.d**. These files are executed in ASCII order (see FILES below for more information), terminating some system service. The combination of commands in **/etc/rc0** and files in **/etc/shutdown.d** and **/etc/rc0.d** determines how the system is shut down.

The recommended sequence for **/etc/rc0** is:

Stop System Services and Daemons.

> Various system services (such as LP Spooler) are gracefully terminated.
>
> When new services are added that should be terminated when the system is shut down, the appropriate files are installed in **/etc/shutdown.d** and **/etc/rc0.d**.

Terminate Processes

> SIGTERM signals are sent to all running processes by *killall*(1M). Processes stop themselves cleanly if sent SIGTERM.

Kill Processes

> SIGKILL signals are sent to all remaining processes; no process can resist SIGKILL.

At this point, the only processes left are those associated with **/etc/rc0** and processes 0 and 1, which are special to the operating system.

Unmount All File Systems

Only the root file system (/) remains mounted.

Depending on which system state the systems end up in (0, 5, or 6), the entries in **/etc/inittab** will direct what happens next. If the **/etc/inittab** has not defined any other actions to be performed as in the case of system state 0, then the operating system will have nothing to do.

FILES

The execution by **/bin/sh** of any files in **/etc/shutdown.d** occurs in ASCII sort-sequence order. See *rc2*(1M) for more information.

SEE ALSO

killall(1M), rc2(1M), shutdown(1M)

BUGS

SYSTEM V/88 cannot be rebooted from software; the reset switch must always be used.

NAME

rc2 – run commands performed for multi-user environment

SYNOPSIS

/etc/rc2

DESCRIPTION

This file is executed via an entry in **/etc/inittab** and is responsible for those initializations that bring the system to a ready-to-use state, traditionally state 2, called the multi-user state.

The actions performed by **/etc/rc2** are found in files in the directory **/etc/rc.d** and files beginning with S in **/etc/rc2.d**. These files are executed by **/bin/sh** in ASCII sort–sequence order (see FILES for more information). When functions are added that need to be initialized when the system goes multi-user, an appropriate file should be added in **/etc/rc2.d**.

The functions done by **/etc/rc2** command and associated **etc/rc2.d** files include:

Setting-up and mounting the user (**/usr**) file system.

Cleaning up (remaking) the **/tmp** and **/usr/tmp** directories.

Loading the network interface and ports cards with program data and starting the associated processes.

Starting the *cron* daemon by executing **/etc/cron**.

Cleaning up (deleting) uucp locks status, and temporary files in the **/usr/spool/uucp** directory.

Other functions can be added, as required, to support the addition of hardware and software features.

EXAMPLES

The following are prototypical files found in **/etc/rc2.d**. These files are prefixed by an S and a number indicating the execution order of the files.

MOUNTFILESYS

```
#    Set up and mount file systems

cd /
/etc/mountall /etc/fstab
```

```
RMTMPFILES
        #  clean up /tmp
        rm –rf /tmp
        mkdir /tmp
        chmod 777 /tmp
        chgrp sys /tmp
        chown sys /tmp

uucp
        #    clean-up uucp locks, status, and temporary files

        rm –rf /usr/spool/locks/*
```

FILES

Here are some hints about files in **/etc/rc.d (/etc/rc.d2.d)**:

The order in which files are executed is important. Since they are executed in ASCII sort–sequence order, using the first character of the file name as a sequence indicator will help keep the proper order. Thus, files starting with the following characters would be:

[0-9].	very early
[A-Z].	early
[a-n].	later
[o-z].	last

Files in **/etc/rc.d** that begin with a dot (.) are not executed. This feature can be used to hide files that are not to be executed for the time being without removing them. The command can be used only by the super-user.

Files in **/etc/rc2.d** must begin with an **S** or a **K** followed by a number and the rest of the file name. Upon entering run level 2, files beginning with **S** are executed with the **start** option; files beginning with **K**, are executed with the **stop** option. Files beginning with other characters are ignored.

SEE ALSO

shutdown(1M)

NAME

rfadmin – Remote File Sharing administration

SYNOPSIS

rfadmin

rfadmin –[ar] *domain.nodename*

rfadmin –[pq]

rfadmin –o *loopback noloopback*

DESCRIPTION

rfadmin is primarily used to add and remove computers and their associated authentication information from a *domain*/**passwd** file on a Remote File Sharing primary domain name server. It is also used to transfer domain name server responsibilities from one machine to another. Used with no options, *rfadmin* returns the *domain.nodename* of the current domain name server for the local domain. Other options let you check if RFS is running and turn on the RFS loop back feature.

rfadmin can only be used to modify domain files on the primary domain name server (**–a** and **–r** options). If domain name server responsibilities are temporarily passed to a secondary domain name server, that computer can use the **–p** option to pass domain name server responsibility back to the primary. *rfadmin* can be used on any computer with no options or with the **q** or **o** options, to print information about the current domain name server. The user must have **root** permissions to use the command.

–a *domain.nodename*
Used to add a computer to the member list of the domain that is served by this primary domain name server. The computer's name must be of the form *domain.nodename*. This command creates an entry for *nodename* in the *domain*/**passwd** file, which has the same format as **/etc/passwd**, and prompts for an initial authentication password. The password prompting process conforms with that of *passwd*(1).

–r *domain.nodename*
Used to remove a computer from its domain by removing it from the *domain*/**passwd** file.

–p
Used to pass the domain name server responsibilities back to a primary or to a secondary name server.

–q
Prints a message that will tell you whether or not RFS is running.

–o *option*
Lets you set RFS system options, by replacing *option* with one of the following:

loopback
Enables loop back facility for your computer. When this is set, you can mount a resource that is advertised from your own computer. This is used for testing applications in RFS when only one computer is available. Loop back is off by default.

noloopback
Turns off the loop back facility for your computer. This is the default.

ERRORS

When used with the **–a** option, if *domain.nodename* is not unique in the domain, an error message will be sent to standard error.

When used with the **–r** option, if (1) *domain.nodename* does not exist in the domain, (2) *domain.nodename* is defined as a domain name server, or (3) there are resources advertised by *domain.nodename*, an error message will be sent to standard error.

When used with the **–p** option to change the domain name server, if there are no backup name servers defined for *domain*, a warning message will be sent to standard error.

FILES

/usr/nserve/auth.info/*domain***/passwd**
(For each *domain*, this file: is created on the primary,
should be copied to all secondaries, and should be copied to
all computers that want to do password verification of computers
in the *domain*.)

SEE ALSO

passwd(1), rfstart(1M), rfstop(1M), umount(1M)

NAME

rfpasswd – change Remote File Sharing host password

SYNOPSIS

rfpasswd

DESCRIPTION

rfpasswd updates the Remote File Sharing authentication password for a host; processing of the new password follows the same criteria as *passwd*(1). The updated password is registered at the domain name server (/usr/nserve/auth.info/*domain*/passwd) and replaces the password stored at the local host (**/usr/nserve/loc.passwd** file).

This command is restricted to the superuser.

NOTE: If you change your host password, make sure that hosts that validate your password are notified of this change. To receive the new password, hosts must obtain a copy of the *domain*/**passwd** file from the domain's primary name server. If this is not done, attempts to mount remote resources may fail!

ERRORS

If (1) the old password entered from this command does not match the existing password for this machine, (2) the two new passwords entered from this command do not match, (3) the new password does not satisfy the security criteria in *passwd*(1), (4) the domain name server does not know about this machine, or (5) the command is not run with super-user privileges, an error message will be sent to standard error. Also, Remote File Sharing must be running on your host and your domain's primary name server. A new password cannot be logged if a secondary is acting as the domain name server.

FILES

/usr/nserve/auth.info/*domain***/passwd**
/usr/nserve/loc.passwd

SEE ALSO

passwd(1), rfstart(1M), rfadmin(1M)

NAME

rfstart – start Remote File Sharing

SYNOPSIS

rfstart [-v] [-p*primary_addr*]

DESCRIPTION

rfstart starts RFS and defines an authentication level for incoming requests. (This command can only be used after the domain name server is set up and your computer's domain name and network specification has been defined using *dname*(1M).)

Specifies that verification of all clients is required in response to initial incoming mount requests; any host not in the file **/usr/nserve/auth.info/***domain*/passwd for the **domain** they belong to, will not be allowed to mount resources from your host.. If **-v** is not specified, hosts named in *domain*/passwd will be verified, other hosts will be allowed to connect without verification.

Indicates the primary domain name server for your domain. *primary_addr* must be the network address of the primary name server for your domain. If the -p option is not specified, the address of the domain name server is taken from the *rfmaster* file. (See *rfmaster*(1M) for a description of the valid address syntax.)

If the host password has not been set, *rfstart* will prompt for a password; the password prompting process must match the password entered for your machine at the primary domain name server (see *rfadmin*(1M)). if you remove the **loc.passwd** file or change domains, you will also have to reenter the password.

Also, when *rfstart* is run on a domain name server, entries in the *rfmaster*(4) file are syntactically validated.

This command is restricted to the superuser.

ERRORS

If syntax errors are found in validating the *rfmaster*(4) file, a warning describing each error will be sent to standard error.

If (1) the shared resource environment is already running, (2) there is no communications network, (3) the domain name server cannot be found, (4) the domain name server does not recognize the machine, or (5) the command is run without super-user privileges, an error message will be sent to standard error.

Remote file sharing will not start if the host password in **/usr/nserve/loc.passwd** is corrupted. If you suspect this has happened, remove the file and run *rfstart* again to reenter your password.

NOTE: *rfstart* will *not* fail if your host password does not match the password on the domain name server. You will simply receive a warning message. However, if you try to mount a resource from the primary or any other host that validates your password, the mount will fail if your password does not match the one that host has listed for your machine.

FILES

/usr/nserve/rfmaster
/usr/nserve/loc.passwd

SEE ALSO

adv(1M), dname(1M), mount(1M), rfadmin(1M), rfstop(1M), unadv(1M)
rfmaster(4) in the *Programmer's Reference Manual*.

NAME

rfstop – stop the Remote File Sharing environment

SYNOPSIS

rfstop

DESCRIPTION

rfstop disconnects a host from the RFS environment until another *rfstart*(1M) is executed.

When executed on the domain name server, the domain name server responsibility is moved to a secondary name server as designated in the *rfmaster*(4) file. If there is no designated secondary name server *rfstop* will issue a warning message, RFS will be stopped, and name service will no longer be available to the domain.

This command is restricted to the superuser.

ERRORS

If (1) there are resources currently advertised by this host, (2) resources from this machine are still remotely mounted by other hosts, (3) there are still remotely mounted resources in the local file system tree, (4) *rfstart*(1M) had not previously been executed, or (5) the command is not run with superuser privileges, an error message will be sent to standard error and RFS will not be stopped.

SEE ALSO

adv(1M), mount(1M), rfadmin(1M), rfstart(1M), unadv(1M), rfmaster(4)

NAME

rfuadmin – Remote File Sharing notification shell script

SYNOPSIS

rfuadmin *message remote_resource* [*seconds*]

DESCRIPTION

The *rfuadmin* administrative shell script responds to unexpected RFS events, e.g., broken network connections and forced unmounts, picked up by the *rfudaemon* process. This command is not intended to be run directly from the shell.

The response to messages received by *rfudaemon* can be tailored to suit the particular system by editing the *rfuadmin* script. The following paragraphs describe the arguments passed to *rfuadmin* and the responses.

disconnect *remote_resource*

A link to a remote resource has been cut. *rfudaemon* executes *rfuadmin*, passing it the message **disconnect** and the name of the disconnected resource. *rfuadmin* sends this message to all terminals using *wall*(1):

Remote_resource `has been disconnected from the system.`

Then it executes *fuser*(1M) to kill all processes using the resource, unmounts the resource (*umount*(1M)) to clean up the kernel, and starts *rmount* to try to remount the resource.

fumount *remote_resource*

A remote server machine has forced an unmount of a resource a local machine has mounted. The processing is similar to processing for a disconnect.

fuwarn *remote_resource seconds*

This message notifies *rfuadmin* that a resource is about to be unmounted. *rfudaemon* sends this script the *fuwarn* message, the resource name, and the number of seconds in which the forced unmount will occur. *rfuadmin* sends this message to all terminals:

Remote_resource `is being removed from the system in` # *seconds*.

SEE ALSO

fumount(1M), rmount(1M), rfudaemon(1M), rfstart(1M)
wall(1) in the *User's Reference Manual*.

BUGS

The console must be on when RFS is running. If it is not, *rfuadmin* will hang when it tries to write to the console (*wall*) and recovery from disconected resources will not complete.

NAME

rfudaemon – Remote File Sharing daemon process

SYNOPSIS

/usr/nserve/rfudaemon

DESCRIPTION

The *rfudaemon* command is started automatically by *rfstart*(1M) and runs as a daemon process as long as RFS is active. Its function is to listen for unexpected events, such as broken network connections and forced unmounts, and execute appropriate administrative procedures.

When such an event occurs, *rfudaemon* executes the administrative shell script *rfuadmin*, with arguments that identify the event. This command is not intended to be run from the shell. Here are the events:

DISCONNECT
A link to a remote resource has been cut. *rfudaemon* executes *rfuadmin*, with two arguments: **disconnect** and the name of the disconnected resource.

FUMOUNT
A remote server machine has forced an unmount of a resource a local machine has mounted. *rfudaemon* executes *rfuadmin*, with two arguments: **fumount** and the name of the disconnected resource.

GETUMSG
A remote user-level program has sent a message to the local *rfudaemon*. Currently the only message sent is *fuwarn*, which notifies *rfuadmin* that a resource is about to be unmounted. It sends *rfuadmin* the *fuwarn*, the resource name, and the number of seconds in which the forced unmount will occur.

LASTUMSG
The local machine wants to stop the *rfudaemon* (*rfstop*(1M)). This causes *rfudaemon* to exit.

SEE ALSO

rfstart(1M), rfuadmin(1M)

NAME

rmntstat – display mounted resource information

SYNOPSIS

rmntstat [**-h**] [*resource*]

DESCRIPTION

When used with no options, *rmntstat* displays a list of all local RFS resources that are remotely mounted, the local path name, and the corresponding clients. *rmntstat* returns the remote mount data regardless of whether a resource is currently advertised; this ensures that resources that have been unadvertised but are still remotely mounted are included in the report. When a *resource* is specified, *rmntstat* displays the remote mount information only for that resource. The *-h* option causes header information to be omitted from the display.

EXIT STATUS

If no local resources are remotely mounted, *rmntstat* will return a successful exit status.

ERRORS

If *resource* (1) does not physically reside on the local machine or (2) is an invalid resource name, an error message will be sent to standard error.

SEE ALSO

mount(1M), fumount(1M), unadv(1M)

NAME

rmnttry – attempt to mount queued remote resources

SYNOPSIS

/usr/nserve/rmnttry [*resource* ...]

DESCRIPTION

rmnttry sequences through the pending mount requests stored in **/usr/nserve/rmnttab**, trying to mount each resource. If a mount succeeds, the resource entry is removed from the **/usr/nserve/rmnttab** file.

If one or more resource names are supplied, mounts are attempted only for those resources, rather than for all pending mounts. Mounts are not attempted for resources not present in the **/usr/nserve/rmnttab** file (see *rmount*(1M)). If a mount invoked from *rmnttry* takes over 3 minutes to complete, *rmnttry* aborts the mount and issues a warning message.

rmnttry is typically invoked from a cron entry in **/usr/spool/cron/crontabs/root** to attempt mounting queued resources at periodic intervals. The default strategy is to attempt mounts at 15 minute intervals. The cron entry for this is:

```
10,25,40,55 * * * * /usr/nserve/rmnttry >/dev/null
```

FILES

/usr/nserve/rmnttab	pending mount requests

SEE ALSO

mount(1M), rmount(1M), rumount(1M), mnttab(4)
crontab(1) in the *User's Reference Manual*.

DIAGNOSTICS

An exit code of 0 is returned if all requested mounts succeeded, 1 is returned if one or more mounts failed, and 2 is returned for bad usage.

NAME

rmount – queue remote resource mounts

SYNOPSIS

/etc/rmount [–d[r] *resource directory*]

DESCRIPTION

rmount queues a remote resource for mounting. The command enters the resource request into **/usr/nserve/rmnttab**, which is formatted identically to *mnttab*(4). *rmnttry*(1M) is used to poll entries in this file.

When used without arguments, *rmount* prints a list of resources with pending mounts along with their destined directories, modes, and date of request. The resources are listed chronologically, with the oldest resource request appearing first.

The following options are available:

–d
indicates that the *resource* is a remote resource to be mounted on *directory*.

–r
indicates that the *resource* is to be mounted read-only. If the *resource* is write-protected, this flag must be used.

FILES

/usr/nserve/rmnttab
pending mount requests

SEE ALSO

mount(1M), rmnttry(1M), rumount(1M), rmountall(1M), mnttab(4)

DIAGNOSTICS

An exit code of 0 is returned upon successful completion of *rmount*. Otherwise, a non-zero value is returned.

NAME

rmountall, rumountall – mount, unmount Remote File Sharing resources

SYNOPSIS

/etc/rmountall [–] " *file-system-table* " [...]
/etc/rumountall [–k]

DESCRIPTION

rmountall is a RFS command used to mount remote resources according to a *file-system-table*. (**/etc/fstab** is the recommended *file-system-table*.) *rmountall* also invokes the */usr/nserve/rmnttry*(1M) command which attempts to mount queued resources. The special file name "-" reads from the standard input.

rumountall causes all mounted remote resources to be unmounted and deletes all resources that were queued from */etc/rmount*(1M). The **–k** option sends a SIGKILL signal, via *fuser*(1M), to processes that have files open.

These commands may be executed only by the superuser.

The file-system-table format is:

column 1	block special file name of file system
column 2	mount-point directory
column 3	–r if to be mounted read-only; –d if remote resource
column 4	file system type (not used with Remote File Sharing)
column 5+	ignored

White-space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

SEE ALSO

fuser(1M), mount(1M), rfstart(1M), rmnttry(1M), rmount(1M),
sysadm(1) in the *User's Reference Manual*.
signal(2) in the *Programmer's Reference Manual*.

DIAGNOSTICS

No messages are printed if the remote resources are mounted successfully.

Error and warning messages come from *mount*(1M).

NAME

rumount – cancel queued remote resource request

SYNOPSIS

/usr/nserve/rumount *resource* ...

DESCRIPTION

rumount cancels a request for one or more resources that are queued for mount. The entries for the resources are deleted from **/usr/nserve/rmnttab**.

FILES

/usr/nserve/rmnttab	pending mount requests

SEE ALSO

mount(1M), rmnttry(1M), rmount(1M), rumountall(1M), mnttab(4)

DIAGNOSTICS

An exit code of 0 is returned if *rumount* completes successfully. A 1 is returned if the resource requested for dequeuing is not in **/usr/nserve/rmnttab**, and a 2 is returned for bad usage or an error in reading or writing **/usr/nserve/rmnttab**.

NAME

sar: sa1, sa2, sadc – system activity report package

SYNOPSIS

/usr/lib/sa/sadc [t n] [ofile]

/usr/lib/sa/sa1 [t n]

/usr/lib/sa/sa2 [**–ubdycwaqvmprSDA**] [**–s** time] [**–e** time] [**–i** sec]

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 counters for CPU utilization, buffer usage, disk and tape I/O activity, TTY device activity, switching and system-call activity, file-access, queue activity, interprocess communications, paging, and RFS.

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, when booting to a multiuser state, to mark the time at which the counters restart from zero. For example, the **/etc/init.d/perf** file writes the restart mark to the daily data by the command entry:

```
su sys –c "/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`"
```

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file **/usr/adm/sa/sa***dd* where *dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted. The following entries in **/usr/spool/cron/crontabs/sys** (see *cron*(1M)) produce records every 20 minutes during working hours, otherwise hourly:

```
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8–17 * * 1–5 /usr/lib/sa/sa1
```

The shell script *sa2*, a variant of *sar*(1), writes a daily report in file **/usr/adm/sa/sar***dd*. The options are explained in *sar*(1). The following **/usr/spool/cron/crontabs/sys** entry reports important activities hourly during the working day:

```
5 18 * * 1–5 /usr/lib/sa/sa2 –s 8:00 –e 18:01 –i 1200 –A
```

The structure of the binary daily data file is:

```
struct sa {
  struct sysinfo si;  /* see /usr/include/sys/sysinfo.h */
  struct minfo mi;  /* defined in sys/sysinfo.h */
  struck dinfo di;  /* RFS info defined in sys/sysinfo.h */
  int minserve,maxserve;/* RFS server low and high water marks */
  int   szinode;  /* current size of inode table  */
  int   szfile;  /* current size of file table  */
  int   szproc;  /* current size of proc table  */
  int   szlckf;  /* current size of file record header table */
  int   szlckr;  /* current size of file record lock table */
  int   mszinode;  /* size of inode table  */
  int   mszfile;  /* size of file table  */
  int   mszproc;  /* size of proc table  */
  int   mszlckf;  /* maximum size of file record header table */
  int   mszlckr;  /* maximum size of file record lock table */
  long  inodeovf;  /* cumulative overflows of inode table  */
  long  fileovf;  /* cumulative overflows of file table  */
  long  procovf;  /* cumulative overflows of proc table  */
  time_t  ts;  /* time stamp, seconds  */
  long  devio[NDEVS][4];      /* device unit information  */
#define IO_OPS   0  /* cumulative I/O requests  */
#define IO_BCNT  1  /* cumulative blocks transferred */
#define IO_ACT   2  /* cumulative drive busy time in ticks */
#define IO_RESP  3  /* cumulative I/O resp time in ticks */
};
```

FILES

/usr/adm/sa/sa*dd*	daily data file
/usr/adm/sa/sar*dd*	daily report file
/tmp/sa.*adrfl*	address file

SEE ALSO

cron(1M)
sar(1), timex(1) in the *User's Reference Manual*.

NAME

savecore – save the core image of a system dump in a file

SYNOPSIS

/etc/savecore [*kernel dumpdirectory dumpfile dumpdev*]

DESCRIPTION

The *savecore* utility is used to save a valid core image (produced at system panic) on the dumpdevice into a normal file which subsequently can be used with *crash*(1M).

The *savecore* program is run automatically at system startup time, and uses **/unix** to determine the dumpdevice. It will save a valid dump into **/usr/adm/kernelcore** as default. *savecore* can also be run by hand, and the user will have to supply from one to four parameters. No parameter can be omitted, however the nullstring "" can be used for any of them.

If there is not enough space in the filesystem in which dumpdir resides savecore will inform the user.

After succesfully saving the core image, the dump on the dumpdevice will be invalidated.

EXAMPLE

The command:

```
/etc/savecore "" /tmp mycore /dev/rdsk/m323_0s1
```

will check whether there is a valid core image on **/dev/rdsk/m323_0s1**. If it finds one and there is enough space in /tmp it will save it in the file **/tmp/mycore**.

FILES

/unix	kernel
/usr/adm/kernelcore	savefile

SEE ALSO

crash(1M), autodump(1M)

DIAGNOSTICS

savecore reports the following conditions:

the dumpdevice for the specified kernel cannot be found

the specified dump directory does not exist

there is not enough disk space to save the core image

(un-)successful completion

NAME

setmnt – establish mount table

SYNOPSIS

/etc/setmnt

DESCRIPTION

setmnt creates the **/etc/mnttab** table which is needed for both the *mount*(1M) and *umount* commands. *setmnt* reads standard input and creates a *mnttab* entry for each line. Input lines have the format:

filesys node

where *filesys* is the name of the file system's *special file* (e.g., **/dev/dsk/***cntlr*_**0s0**) and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the mount table entry.

FILES

/etc/mnttab

SEE ALSO

mount(1M)

BUGS

Problems may occur if *filesys* or *node* are longer than 32 characters.
setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.

NAME

shutdown – shut down system, change system state

SYNOPSIS

/etc/shutdown [**–y**] [**–g***grace_period* [**–i***init_state*]

DESCRIPTION

This command is executed by the superuser to change the state of the machine. By default, it brings the system to a state where only the console has access to the system. This state is traditionally called single-user.

The command sends a warning message and a final message before it starts actual shutdown activities. By default, the command asks for confirmation before it starts shutting down daemons and killing processes. The options are used as follows:

–y
pre-answers the confirmation question so the command can be run without user intervention. A default of 60 seconds is allowed between the warning message and the final message. Another 60 seconds is allowed between the final message and the confirmation.

–g*grace_period*
allows the superuser to change the number of seconds from the 60-second default.

–i*init_state*
specifies the state that *init*(1M) is to be put in following the warnings, if any. By default, system state "**s**" is used.

Recommended system state definitions are:

state 0
Shut the machine down so it is safe to remove the power. Have the machine remove power if it can. The **/etc/rc0** procedure is called to do this work.

state s, S
Bring the machine to the state traditionally called single-user. The **/etc/rc0** procedure is called to do this work. These states unmount all file systems except for the root file system. They kill all processes except for special operating system related processes.

state 5,6
The system is secured for powerdown/reset.

SEE ALSO

init(1M), rc0(1M), rc2(1M)
inittab(4) in the *Programmer's Reference Manual*.

NAME

sledit, msledit – slice table editor

SYNOPSIS

sledit [**–q –n –r –v –l –s** *file*] *rawdev*
msledit [**–c –d** *file* **–D** *dir* **–r –u –e** *file* **–o** *file* **–m –p –s** *dev* **–v –w**] *rawdev*

DESCRIPTION

sledit allows editing of the slice table, sizes and offsets, including the size, logical block size, name, and the volume id of the file system. *msledit* is the stripped down version of *sledit* that allows editing of the slice table without the **usr** file system mounted. The following options are interpreted by *sledit*:

–q	Quiet warning messages.
–n	Non-interactive mode (used with the -s option).
–v	Questions are asked when making a file system.
–r	The slice table is accessed read-only.
–l	Allow editing of the last slice (the slice representing the entire disk).
–s	The slice table is taken from *file* where *file* is either another disk device with a slice table or a file that was copied from another disk device with a slice table.

The following are *msledit* options:

–c	Check for slice table.
–d *filename*	Create slicetable using "ddefs" filename.
–D *dir*	Use "ddefs" directory "dir".
–r	Make "root" only slice table from "ddefs" entry.
–u	Make "root" and "usr" slice table from "ddefs" entry.
–e file	Input slice table from ASCII file.
–o file	Output slice table to ASCII file.
–m	Do not make file systems.
–p	Print slice table to stdout.

–s dev Slice table from device.

–v Ask questions during **mkfs**.

–w Write slice table to device (no edit).

The *rawdev* argument specifies the raw disk device. *sledit* must be used on the slice representing the entire raw disk; e.g., slice 7, slice 15, slice 31. The name of the raw device must be in the form **/dev/rdsk/***controller_X*s*Y* where *X* denotes the device number and *Y* denotes the slice number. For example:

```
sledit /dev/rdsk/m320_0s7
```

In addition, the raw device must have a corresponding block device, **/dev/dsk/***controller_X*s*Y*, with the same minor number as the raw device.

sledit checks the disk device being edited for mounted file systems. If a file system is mounted, then the slice where the file system resides will be marked as read-only.

msledit uses the same method as *ddefs*(1M) for editing the slice table. Each slice is broken into a series of lines containing information about the slice. To modify the information, move the cursor to the appropriate line and type the new value. To obtain information about commands, type help on any line; for detailed information about a specific line, type a question mark (?). To print the slice table, type **t**.

sledit uses a subset of *vi*(1) commands for cursor movement and file manipulation. The following movement commands are interpreted by *sledit:*

Command	Movement
h or ^H	left
l or ^L	right
j or ^J	down
k or ^K	up
arrow keys	same as hljk
w	word forward
b	word backward
e	end of word

The cursor is allowed to move only in and to the predefined fields on the table.

The following commands allow changing the slice table:

Command	Manipulation
:s *file*	save slice table in *file*
^F	next screen
^B	previous screen
O	open a new slice
C	compress two slices
:q	quit
ZZ	quit
:q!	quit (no disk write)
:cw	change word
:dw	delete word
i	insert before
a	append after
ESC	end change
x	delete single char
r	replace single char
.	insert last inserted string
u	undo last change
:!	shell escape

The size and offset fields have a ten-character maximum length, and the file system name and volume id fields have a six-character maximum length. The logical block size field has a length of one. Only decimal values are allowed in the size, offset, and logical block size fields.

The offset and size fields for the slice table and file systems are in units of blocks, where blocks are 512 bytes. The logical block size field is in units of 1K (1024) bytes. Valid entries are 1, 2, 4, or 8.

Eight slices are displayed at one time. If the disk device has been configured with more slices (see *dinit*(1M)), then more screens may be edited.

The :s *file* command saves the slice table, as it appears on the screen, to the regular file *file*. This command is handy for saving a slice table prior to editing or creating data files for the –s option.

The **O** (open) command allows easy allocation of the next non-overlapped slice. Check to make sure there is a previous slice with a slice size greater than its file system size. Then position the cursor in the slice to be opened. The size of the previous slice is reduced to its file system size and the remaining space is given to the new slice.

The **C** (compress) command salvages unused space. Check to make sure that the previous slices are non-overlapped and continuous. Then position the cursor in the slice that is to be compressed. The file system (if any) is destroyed and the space salvaged is given to a previous slice. The last slice cannot be compressed.

sledit writes to disk only when doing a quit, **:q** or **ZZ**. A **:q!** leaves the disk untouched.

sledit uses *curses*(3X) to provide the editing capabilities. To produce terminal dependent output, the type of terminal being used has to be specified. The usual convention for doing this is setting the *TERM* variable in the shell's environment (see *sh*(1)). If the terminal is an EXORterm 155, then the *TERM* could be set to **155**. If the environment variable *TERMINFO* is set, then a local terminal definition will be checked before checking in the standard place, **/usr/lib/terminfo/1/155**. A local definition of the terminal will allow *sledit* to edit the **usr** file system.

If the size of a file system is changed or the offset of the slice where a file system is located is changed, *sledit* and *msledit* will call *mkfs*(1M) on quitting to make file systems on each changed slice. If the **-v** option is specified on the command line, then the number of inodes, gap size, and blocks per cycle can be altered; otherwise, the default values are used.

The **-s** option can be used to copy a slice table from another disk device or a data file. Data files may be created using the **:s** *file* command (with *sledit*) or the **w** *file* command (with *msledit*).

The **–n** option may be used with the **–s** option to simply copy a slice table without using the editor.

DIAGNOSTICS

sledit and *msledit* issue various diagnostic messages. Some of them are concerned with the editing process and are self-explanatory.

Message	Possible Cause
Can't open *xxx*	File *xxx* inaccessible, either denied access permissions or file doesn't exist.
Can't split field	Attempt was made to insert white space while inserting text with **i**, **a** or **r** commands.
File system size greater than slice size.	Making file system size larger than slice.
Missing device file	Raw device not given
No slice table	Disk not formatted with slice table; see *dinit*(1M).
Only one screen	Slice table has only 8 slices; see *dinit*(1M).
Values are in decimal	Inserting non-digit in the size or offset fields.
Slice beyond end of disk	Slice size is larger than the end of the disk.
Slice table ioctl error	Disk driver unable to change slicing.
Slice table read error	Can't read slice table, possibly corrupted.
Slice table seek error	Volume id header corrupted; slice table location wrong.
Super block read error	Slice table corrupted; file system offset wrong.

DIAGNOSTICS (Cont'd.)

Message	**Possible Cause**
Unknown or missing field	Size or offset field blank, if empty must be **0**.
No space for slice table	*malloc*(3X) fails.
Can't compress slice 0	Trying to compress while on slice 0.
Can't stat *xxx*	Same as "Can't open". File *xxx* inaccessible, either denied access permissions or file doesn't exist.
Filename missing	Using ":s" command without filename.
Not a regular file: *xxx*	File *xxx* is a special file.
No slices to compress with.	Previous slices zero length.
Can't compress, no size	Trying to compress a zero length slice.
Can't compress, overlapped	Trying to compress to a slice that is overlapped with the slice being compressed.
Can't compress, non-continuous slices	Slice being compressed is not equal to slice offset of previous slice.
Last slice unavailable	Trying to allocate space from the last slice.
No available space to allocate	Last previous slice has no unused space remaining.
Previous slice has size but no file system	*sledit* cannot allocate space from a slice unless it knows the size of the file system.

DIAGNOSTICS (Cont'd.)

Message	**Possible Cause**
Unknown TERM, TERMINFO or can't find terminfo file.	Environment variables not set properly or the terminfo file for terminal cannot be found.
Volume id read error	Volume id header corrupted or *sledit* being used on a slice not representing entire disk; e.g., slice 7.
Slice offset overlap	Two slices overlap.
Mounted file system, Read only	The slice that is being changed has a mounted file system. Unmount the file system to change.
Must be 1, 2, 4, or 8.	The logical block size may be set to 1, 2, 4, or 8K.

WARNINGS

Use of the **-l** option to edit the slice representing the entire disk is dangerous. Changing the slice offset or the size may prevent the device driver from opening that device again. If that should occur, shut down and reboot the system to solve the problem. If the problem persists, reformat the device.

sledit allows the entire disk to be used for slicing, including those areas reserved for other uses (refer to *ddefs*(1M)). If these areas are allocated by *sledit*, unpredictable results may occur.

On some installations, slice 0 contains the swap area. If this is the case, do not allocate this space to another slice.

sledit allows slices to be overlapped. If this is done, care must be taken not to overlap the file systems on those slices or unpredictable results may occur.

SEE ALSO

mkfs(1M), dinit(1M), ddefs(1M).
dd(1) in the *SYSTEM V/68 User's Reference Manual.*
curses(3X) in the *SYSTEM V/68 Programmer's Reference Manual.*

NAME

strace – print STREAMS trace messages

SYNOPSIS

strace [*mid sid level*] ...

DESCRIPTION

strace without arguments writes all STREAMS event trace messages from all drivers and modules to its standard output. These messages are obtained from the STREAMS log driver (*log*(7)). If arguments are provided they must be in triplets of the form *mid*, *sid*, *level*, where *mid* is a STREAMS module id number, *sid* is a sub-id number, and *level* is a tracing priority level. Each triplet indicates that tracing messages are to be received from the given module/driver, sub-id (usually indicating minor device), and priority level equal to or less than the given level. The token *all* may be used for any member to indicate no restriction for that attribute.

The format of each trace message output is:

<seq> <time> <ticks> <level> <flags> <mid> <sid> <text>

<seq>	trace sequence number
<time>	time of message in hh:mm:ss
<ticks>	time of message in machine ticks since boot
<level>	tracing priority level
<flags>	E : message is also in the error log F : indicates a fatal error N : mail was sent to the system administrator
<mid>	module id number of source
<sid>	sub-id number of source
<text>	formatted text of the trace message

Once initiated, *strace* will continue to execute until terminated by the user.

EXAMPLES

Output all trace messages from the module or driver whose module id is 41:

```
strace 41 all all
```

Output those trace messages from driver/module id 41 with sub-ids 0, 1, or 2:

```
strace  41 0 1  41 1 1  41 2 0
```

Messages from sub-ids 0 and 1 must have a tracing level less than or equal to 1. Those from sub-id 2 must have a tracing level of 0.

WARNINGS

Due to performance considerations, only one *strace* process is permitted to open the STREAMS log driver at a time. The log driver has a list of the triplets specified in the command invocation, and compares each potential trace message against this list to decide if it should be formatted and sent up to the *strace* process. Hence, long lists of triplets will have a greater impact on overall STREAMS performance. Running *strace* will have the most impact on the timing of the modules and drivers generating the trace messages that are sent to the *strace* process. If trace messages are generated faster than the *strace* process can handle them, then some of the messages will be lost. This last case can be determined by examining the sequence numbers on the trace messages output.

SEE ALSO

log(7)
STREAMS Programmer's Guide.

NAME

strclean – STREAMS error logger cleanup program

SYNOPSIS

strclean [**-d** *logdir*] [**-a** *age*]

DESCRIPTION

strclean is used to clean up the STREAMS error logger directory on a regular basis (for example, by using *cron*(1M)). By default, all files with names matching **error.*** in **/usr/adm/streams** that have not been modified in the last 3 days are removed. A directory other than **/usr/adm/streams** can be specified using the **-d** option. The maximum age in days for a log file can be changed using the **-a** option.

EXAMPLE

```
strclean -d /usr/adm/streams -a 3
```

has the same result as running strclean with no arguments.

NOTES

strclean is typically run from *cron*(1M) on a daily or weekly basis.

FILES

/usr/adm/streams/error.*

SEE ALSO

cron(1M), strerr(1M)
STREAMS Programmer's Guide.

NAME

strerr – STREAMS error logger daemon

SYNOPSIS

strerr

DESCRIPTION

strerr receives error log messages from the STREAMS log driver (*log*(7)) and appends them to a log file. The error log files produced reside in the directory **/usr/adm/streams**, and are named **error.***mm-dd*, where *mm* is the month and *dd* is the day of the messages contained in each log file.

The format of an error log message is:

<seq> <time> <ticks> <flags> <mid> <sid> <text>

<seq>	error sequence number
<time>	time of message in hh:mm:ss
<ticks>	time of message in machine ticks since boot priority level
<flags>	T : the message was also sent to a tracing process F : indicates a fatal error N : send mail to the system administrator
<mid>	module id number of source
<sid>	sub-id number of source
<text>	formatted text of the error message

Messages that appear in the error log are intended to report exceptional conditions that require the attention of the system administrator. Those messages which indicate the total failure of a STREAMS driver or module should have the F flag set. Those messages requiring the immediate attention of the administrator will have the N flag set, which causes the error logger to send the message to the system administrator via *mail*(1). The priority level usually has no meaning in the error log but will have meaning if the message is also sent to a tracer process.

Once initiated, *strerr* will continue to execute until terminated by the user. Commonly, *strerr* would be executed asynchronously.

CAVEATS

Only one *strerr* process at a time is permitted to open the STREAMS log driver.

If a module or driver is generating a large number of error messages, running the error logger will cause a degradation in STREAMS performance. If a large burst of messages are generated in a short time, the log driver may not be able to deliver some of the messages. This situation is indicated by gaps in the sequence numbering of the messages in the log files.

FILES

/usr/adm/streams/error.*mm-dd*

SEE ALSO

log(7)
STREAMS Programmer's Guide.

NAME

su – become super-user or another user

SYNOPSIS

su [**–**] [*name* [*arg* **-c** *string* **-r**]]

DESCRIPTION

su allows one to become another user without logging off. The default user *name* is **root** (i.e., superuser).

To use *su*, the appropriate password must be supplied (unless you are already **root**). If the password is correct, *su* will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry (see *passwd*(4)), or **/bin/sh** if none is specified (see *sh*(1)). To restore normal user ID privileges, type an **EOF** (`cntrl-d`) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like *sh*(1), an *arg* of the form **–c** *string* executes *string* via the shell and an *arg* of **–r** will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(1). If the first argument to *su* is a **–**, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is **–**, thus, causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of **$PATH,** which is set to **/bin:/etc:/usr/bin** for **root**. Note that if the optional program used as the shell is **/bin/sh,** the user's **.profile** can check *arg0* for **–sh** or **–su** to determine if it was invoked by *login*(1) or *su*(1), respectively. If the user's program is other than **/bin/sh,** then **.profile** is invoked with an *arg0* of **-***program* by both *login*(1) and *su*(1).

All attempts to become another user using *su* are logged in the log file **/usr/adm/sulog**.

EXAMPLES

To become user **bin** while retaining your previously exported environment, execute:

```
su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, execute:

```
su - bin
```

To execute *command* with the temporary environment and permissions of user **bin**, type:

su - bin -c "*command args*"

FILES

/etc/passwd	system's password file
/etc/profile	system's profile
/etc/group	system's group file
$HOME/.profile	user's profile
/usr/adm/sulog	log file

SEE ALSO

env(1), login(1), sh(1) in the *User's Reference Manual*.
passwd(4), profile(4), environ(5) in the *Programmer's Reference Manual*.

NAME

swap – swap administrative interface

SYNOPSIS

/etc/swap –a *swapdev swaplow swaplen*
/etc/swap –d *swapdev swaplow*
/etc/swap –l

DESCRIPTION

swap provides a method of adding, deleting, and monitoring the system swap areas used by the memory manager. The following options are recognized:

–a
Add the specified swap area. *swapdev* is the name of the block special device, e.g., **/dev/dsk/m323_0s7**. *swaplow* is the offset in 512-byte blocks into the device where the swap area should begin. *swaplen* is the length of the swap area in 512-byte blocks. This option can only be used by the super-user. Swap areas are normally added by the system start up routine **/etc/rc** when going into multi-user mode.

–d
Delete the specified swap area. *swapdev* is the name of block special device, e.g., **/dev/dsk/m323_0s7**. *swaplow* is the offset in 512-byte blocks into the device where the swap area should begin. Using this option marks the swap area as "INDEL" (in process of being deleted). The system will not allocate any new blocks from the area, and will try to free swap blocks from it. The area will remain in use until all blocks from it are freed. This option can only be used by the super-user.

–l
List the status of all the swap areas. The output has four columns:

DEV
The *swapdev* special file for the swap area if one can be found in the **/dev/dsk** or **/dev** directories, and its major/minor device number in decimal.

LOW
The *swaplow* value for the area in 512-byte blocks.

LEN
The *swaplen* value for the area in 512-byte blocks.

FREE
The number of free 512-byte blocks in the area. If the swap area is being deleted, this column will be marked INDEL.

WARNINGS

No check is done to see if a swap area being added overlaps with an existing swap area or file system.

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync*(2) for details.

NOTE

If you have done a write to a file on a remote machine in a Remote File Sharing environment, you cannot use *sync* to force buffers to be written out to disk on the remote machine. *sync* will only write local buffers to local disks.

SEE ALSO

sync(2) in the *Programmer's Reference Manual*.

NAME

sysadm – menu interface to do system administration

SYNOPSIS

sysadm [*subcommand*]

DESCRIPTION

This command, when invoked without an argument, presents a menu of system administration subcommands, from which the user selects. If the optional argument is presented, the named subcommand is run or the named submenu is presented.

The *sysadm* command may be given a password. See **admpasswd** in the SUBCOMMANDS section.

SUBCOMMANDS

The following menus of subcommands are available. (The number of bullets (●) in front of each item indicates the level of the menu or subcommand.)

● diagnostics

system diagnostics menu

These subcommands look for and sometimes repair problems in the system. Those subcommands that issue reports allow you to determine if there are detectable problems. Commands that attempt repair are for repair people only. You must know what you are doing!

● ● diskrepair

advice on repair of hard disk errors

This subcommand advises you on how to go about repairing errors that occur on hard disks.

WARNING

BECAUSE THIS IS A REPAIR FUNCTION, IT SHOULD ONLY BE PERFORMED BY QUALIFIED SERVICE PERSONNEL.

NOTE: Reports of disk errors frequently result in the loss of files and/or damage to data. It may be necessary to restore some or all of the repaired disk from backup copies.

● ● ● badtracks
Badtracks provides for dynamically redirecting disk tracks which have developed defects since installation. It advises of any files corrupted and repairs any file system damage resulting from the redirection.

● ● diskreport
report on hard disk errors

This subcommand shows you if the system has collected any information indicating that there have been errors while reading the hard disks or diskettes. You can request either summary or full reports. The summary report provides sufficient information about disk errors to determine if repair should be attempted. If a number of errors is reported, there is damage and you should call for service. The full report gives additional detail for the expert repair person trouble shooting complicated problems.

NOTE: Reports of disk errors frequently result in the loss of files and/or damage to data. It may be necessary to restore portions of the repaired disk from backup copies.

● diskmgmt
disk management menu

The subcommands in this menu provide functions for using removable disks. The subcommands include the ability to format disks, copy disks, and to use disks as mountable file systems.

● ● checkfsys
check a removable disk file system for errors

This subcommand checks a file system on a removable disk for errors. If there are errors, this procedure attempts to repair them.

● ● cpdisk
make exact copies of a removable disk

This procedure copies the contents of a removable disk into the machine and then allows the user to make exact copies of it. These copies are identical to the original in every way. The copies are made by first reading the original removable disk entirely into the machine and then writing it out onto duplicate disks. The procedure will fail if there is not enough space in the system to hold the original disk.

● ● erase

erase data from removable disk

This procedure erases a removable disk by overwriting it with null bytes. The main purpose is to remove data that the user does not want seen. Once performed, this operation is irreversible.

● ● format

format new removable disks

This subcommand prepares new removable disks for use. Once formatted, programs and data can be written on the disks.

● ● makefsys

create a new file system on a removable disk

This subcommand creates a new file system on a removable disk which can then store data which the user does not wish to keep on the hard disk. When "mounted", the file system has all the properties of a file system kept on the hard disk, except that it is smaller.

● ● mountfsys

mount a removable disk file system

mountfsys mounts a file system found on a removable disk, making it available to the user. The file system is unmounted with the **umountfsys** command. THE DISK MUST NOT BE REMOVED WHILE THE FILE SYSTEM IS STILL MOUNTED. IF THE FILE SYSTEM HAS BEEN MOUNTED WITH THE **mountfsys** COMMAND, IT MUST BE UNMOUNTED WITH **umountfsys**.

● ● umountfsys

unmount a removable disk file system

umountfsys unmounts a file system, allowing the user to remove the disk. THE DISK MUST NOT BE REMOVED UNTIL THE FILE SYSTEM IS UNMOUNTED. **umountfsys** MAY ONLY BE USED TO UNMOUNT FILE SYSTEMS MOUNTED WITH THE **mountfsys** COMMAND.

● filemgmt

file management menu

The subcommands in this menu allow the user to protect files on the hard disk file systems by copying them onto removable media and later restoring them to the hard disk by copying them back. Subcommands are also provided to determine which files might be best kept on a removable archive based on age or size.

● ● backup_restore

back up files from integral hard disk to removable disk or tape; restore files from removable media to hard disk.

The backup option of this subcommand saves copies of files from the integral hard disk file systems to removable disk or tape. There are three kinds of backups:

FULL - copies all files (useful in case of serious file system damage)

INCREMENTAL - copies files changed since the last full backup

SELECTED - copies a user-defined subset of files

The normal usage is to do a full backup of each file system and then periodically do incremental backups. Two cycles are recommended (one set of full backups and several incrementals to each cycle).

The restore option copies backed up files from disks or tapes onto the hard disk. You can restore individual files, directories of files, or the entire contents of a disk or tape. The user can restore from both "incremental" and "full" media. The user can also list the names of files stored on the disk or tape.

● ● diskuse

display how much of the hard disk is being used

This subcommand lets the user know what percentage of the hard disk is currently occupied by files. The list is organized by file system names.

● ● fileage

list files older than a particular date

This subcommand prints the names of all files older than the date specified by the user.

If no date is entered, all files older than 90 days will be listed. If no directory is specified to look in, the **/usr/admin** directory will be used.

● ● filesize

list the largest files in a particular directory

This subcommand prints the names of the largest files in a specific directory. If no directory is specified, the **/usr/admin** directory will be used. If the user does not specify how many large files to list, 10 files will be listed.

● machinemgmt

machine management menu

Machine management functions are tools used to operate the machine, e.g., turn it off, reboot, or go to the firmware monitor.

● ● firmware

stop all running programs then enter firmware mode

This procedure will stop all running programs, close any open files, write out information to the disk (such as directory information), then enter the firmware mode. (Machine diagnostics and other special functions that are not available from SYSTEM V/88.) Not all machines can fully support this function.

● ● powerdown

stop all running programs, then turn off the machine

This subcommand will stop all running programs, close any open files, write out information to disk (such as directory information), then turn the machine power off. Not all machines can fully support this function.

● ● reboot

stop all running programs then reboot the machine

This subcommand will stop all running programs, close any open files, write out information to disk (such as directory information), then reboot the machine. This can be used to get out of some types of system trouble, such as when a process cannot be killed. Not all machines can fully support this function.

● ● whoson

print list of users currently logged onto the system

This subcommand prints the login ID, terminal device number, and sign-on time of all users who are currently using the computer.

● packagemgmt

package management

These submenus and subcommands manage various software and hardware packages that you install on your machine. Not all optional packages add subcommands here.

- softwaremgmt

  software management menu

  These subcommands permit the user to install new software and remove software. The remove capability is dependent on the particular software package. See the instructions delivered with each package.

- • installpkg

  install new software package onto integral hard disk

  This subcommand copies files from removable disk or tape onto the hard disk and performs additional work if necessary so that the software can be run. From then on, the user will have access to those commands.

- • listpkg

  list packages already installed

  This subcommand shows a list of currently installed optional software packages.

- • removepkg

  remove previously installed package from integral hard disk

  This subcommand displays a list of currently installed optional software packages. Actions necessary to remove the software packages specified by the user will then be performed. The media used to "installpkg" the software may be needed to remove it.

- syssetup

  system setup menu

  System setup routines allow the user to tell the computer what its environment looks like: what the date, time, and time zone is, what administration and system capabilities are to be under password control, what the machine's name is. The first-time setup sequence is also here.

- • admpasswd

  assign or change administrative passwords

  This subcommand lets you set or make changes to passwords for administrative commands and logins such as setup and sysadm.

- • datetime

  set the date, time, time zone, and daylight savings time

This subcommand tells the computer the date, time, time zone, and whether you observe Daylight Savings Time (DST). It is normally run once when the machine is first set up. If you observe DST, the computer will automatically start to observe it in the spring and return to Standard Time in the fall. Setting the time zone and/or setting the calendar date ahead one or more days will cause the *cron*(1) daemon to be restarted.

● ● nodename
set the node name of this machine

This subcommand allows you to change the node name of this machine. The node name is used by various communications networks to identify this machine.

● ● setup
set up your machine the very first time

This subcommand allows the user to define the first login, to set the passwords on the user-definable administration logins and to set the time zone for your location.

● ● syspasswd
assign system passwords

This subcommand lets the user set system passwords normally reserved for the very knowledgeable user. For this reason, this procedure may assign those passwords, but may not change or clear them. Once set, they may only be changed by the specific login or the "root" login.

● ttymgmt
terminal management

This procedure allows the user to manage the computer's terminal functions.

● ● lineset
show TTY line settings and hunt sequences

The TTY line settings are often hunt sequences where, if the first line setting does not work, the line "hunts" to the next line setting until one that does work comes by. This subcommand shows the various sequences with only specific line settings in them. It also shows each line setting in detail.

● ● mklineset
create new tyy line settings and hunt sequences

This subcommand helps you to create TTY line setting entries. You might want to add line settings that are not in the current set or create hunt sequences with only specific line settings in them. The created hunt sequences are circular; stepping past the last setting puts you on the first.

● ● modtty
show and optionally modify characteristics of TTY lines

This subcommand reports and allows you to change the characteristics of TTY lines (also called "ports").

● usermgmt
user management menu

These subcommands allow you to add, modify and delete the list of users that have access to your machine. You can also place them in separate groups so that they can share access to files within the group but protect themselves from other groups.

● ● addgroup
add a group to the system

This subcommand adds a new group name or ID to the computer. Group names and IDs are used to identify groups of users who desire common access to a set of files and directories.

● ● adduser
add a user to the system

This subcommand installs a new login ID on the machine. You are asked a series of questions about the user and then the new entry is made. You can enter more than one user at a time. Once this procedure is finished, the new login ID is available.

● ● delgroup
delete a group from the system

This subcommand allows you to remove groups from the computer. The deleted group is no longer identified by name. However, files may still be identified with the group ID number.

● ● deluser
delete a user from the system

This subcommand allows you to remove users from the computer. The deleted user's files are removed from the hard disk and their logins are removed from the **/etc/passwd** file.

● ● lsgroup
list groups in the system

This subcommand will list all the groups that have been entered into the computer. This list is updated automatically by "addgroup" and "delgroup"

● ● lsuser
list users in the system

This subcommand will list all the users that have been entered into the computer. This list is updated automatically by "adduser" and "deluser".

● ● modadduser
modify defaults used by adduser

This subcommand allows the user to change some of the defaults used when adduser creates a new login. Changing the defaults does not effect any existing logins, only logins made from this point on.

● ● modgroup
make changes to a group on the system

This subcommand allows the user to change the name of a group that the user enters when "addgroup" is run to set up new groups.

● ● moduser
menu of commands to modify a user's login

This menu contains commands that modify the various aspects of a user's login.

● ● ● chgloginid
change a user's login ID

This procedure allows the user to change a user's login ID. Administrative and system logins cannot be changed.

● ● ● chgpasswd
change a user's passwd

This procedure allows removal or change of a user's password. Administrative and system login passwords channot be changed. To change administrative and system login passwords, see the system setup menu: sysadm syssetup.

● ● ● chgshell
change a user's login shell

This procedure allows the user to change the command run when a user logs in. The login shell of the administrative and system logins cannot be changed by this procedure.

● devsetup
tape and floppy device setup menu

This procedure allows the user to change the **ctape, diskette1**, and **hddiskette1** device nodes found in the **/dev/rSA** and **/dev/SA** directories.

● ● changeflop
change **diskette1** and **ddiskette1**

This procedure allows the user to change the **diskette1** and **hddiskette1** device nodes found in the **/dev/rSA** and **/dev/SA** directories. It is suggested that the type of floppy controller and type of floppy is known **before** changing nodes.

● ● changetape
change **ctape**

This procedure allows the user to change the **ctape** device nodes found in the **/dev/rSA** and **/dev/SA** directories. It is suggested that the type of tape controller and type of tape is known **before** changing nodes.

● ● listdevs
This procedure allows the user to list all the devices linked to **/dev/rSA/diskette1, /dev/rSA/hddiskette1**, and **/dev/rSA/ctape**.

EXAMPLES

sysadm adduser

FILES

The files that support *sysadm* are found in **/usr/admin**.
The menus are subdirectories of **/usr/admin/menu**.

NAME

sysdef – output system definition

SYNOPSIS

/etc/sysdef [*system_namelist* [*master*]]

DESCRIPTION

sysdef outputs the current system definition in tabular form. It lists all hardware devices, their local bus addresses, and unit count, as well as pseudo devices, system devices, and the values of all tunable parameters. It generates the output by analyzing the named operating system file (*system_namelist*) and extracting the configuration information from the name list itself.

FILES

/unix	default operating system file (where the system namelist is)

SEE ALSO

master(4), nlist(3C) in the *Programmer's Reference Manual*.
config(1M)

DIAGNOSTICS

`internal name list overflow`

If the master table contains more than an internally specified number of entries for use by *nlist*(3C).

NAME

sysgen – configure and generate the operating system

SYNOPSIS

Interactive:
sysgen [**–l** *path*] [*configuration*]
sysgen [**–l** *path*] **–d** [*collection*]

Non-interactive:
sysgen [**–l** *path*] **–gbi** [*configuration*]
sysgen [**–l** *path*] [**–s/D** *collection*] ... [**–c** *comment*] [*configuration*] ...

DESCRIPTION

sysgen is used to select and make modifications to the system configuration and to create and install an operating system based on the specified configuration. System configuration modifications may be software changes (i.e., tuning the operating system), hardware changes (e.g., upgrades, adding/removing peripherals), or the selection of one configuration from a list of two or more distinct configurations available for the system (e.g., selecting either a configuration with/without network services). *sysgen* operates in both interactive and non-interactive modes.

Interactive

If no options are specified on the command line, *sysgen* displays the configuration screen and permits interactive modification and creation of system configurations (see Procedure 6.1 in the *System Administrator's Guide*).

If a configuration or collection name is specified on the command line, *sysgen* displays the collection or item screen associated with the specified name.

The following options can be used in the interactive mode:

–l
Specify a path to the *sysgen* library where configuration and description information is located. This option permits the user to override the default value that is specified in the *sh*(1) environment variable, SG_LIB. If this variable is not defined, the *sysgen* library path defaults to **/usr/src/uts/mot/sysgen.**

–d
Display the collection screen to permit interactive modification and creation of *sysgen* description information.

When exiting from an interactive *sysgen* session, the following prompt displays for each configuration or collection that has been modified:

`Save changes to` *keyword*? **`[y/n]`**

Type **n** (no) to discard modifications. Type **y** (yes) to save modifications made during *sysgen* execution. After saving changes to the current configuration (i.e., the configuration with the → cursor in the leftmost column), the following prompt appears:

`Update system configuration files? [y/n]`

The system configuration files are data files generated by *sysgen* before generating a new operating system. When you change the current configuration, (by modifying configuration information or selecting a different configuration), you must regenerate the system configuration files to reflect the change. Type **n** to prevent an update to the system configuration files and exit **sysgen**. Type **y** to cause the system configuration files to reflect the new current configuration; the following prompt appears:

`Rebuild the operating system? [y/n]`

Type **n** to prevent rebuilding of the operating system and to exit sysgen. Type y to recreate the operating system; the following prompt appears:

`Install the new operating system to be used on the next reboot? [y/n]`

Type **n** to prevent an update to the operating system. Type y to cause the operating system that was created last to be installed on the next system reboot.

Non-interactive

Non-interactive *sysgen* processing accepts the following options:

–g
Generate system configuration files based on the current configuration that may be specified on the command line. If not specified on the command line, the configuration is taken from the *sgdata* file. This control file usually contains the name of the configuration that was used last. It is updated when the user answers **y** to the **`Update system configuration files?`** question at the end of an interactive sysgen session or when the *sysgen -g* option is invoked non-interactively.

–b
Rebuild the operating system.

–i
Install the operating system for use on the next reboot.

–s
Enable a collection in the current configuration (i.e., configuration specified on the *sysgen* command line or the last configuration selected for the system), then generate the system configuration files, rebuild operating system, and install it for use on the next reboot.

–D
Disable a collection.

–c
Change comment associated with the configuration specified.

The **–g, –b,** and **–i** options are executed in that order but may be specified in any order on the command line.

SCREENS

sysgen screens appear in a hierarchy:

configuration screen
collection screen
item screen
data screen

See Procedure 6.1 in the *System Administrator's Guide* for screen examples.

Invoking *sysgen* with no options causes the configuration screen to be displayed listing the various configurations available for the system. Each line represents one configuration and consists of two fields:

description	shows the configuration associated with the line.
keyword	uniquely identifies the configuration

within *sysgen*. When specifying a configuration argument on the *sysgen* command line (**–g, –s** or no option), use the configuration's *keyword* to identify the configuration. When the configuration screen is displayed, the → symbol points to the current configuration.

Each configuration represented on the configuration screen has a collection screen associated with it. To display a collection screen move the → symbol to the line with the desired configuration and type **o**. Each line on the collection screen represents a collection of *sysgen* descriptions and consists of two fields:

description	specifies the collection's contents.
keyword	uniquely identifies the collection within *sysgen*.

When specifying a collection argument on the *sysgen* command line (**–d** or **–s** option), use the collection's *keyword*. An asterisk (*) in the leftmost column of a given line of the collection screen (except with the **–d** option) indicates that one or more items in the collection have been enabled for the configuration.

An equal sign (=) in the leftmost column of a given line of the collection screen (except with the **-d** option) indicates that this collection is enabled and cannot be disabled.

When a description item is enabled, the information it contains is used to create a new operating system.

Each collection found on the collection screen has an item screen associated with it. To display the item screen move the → symbol to the line with the desired collection and type **o**. Each line of the item screen represents an item of configuration information. There are three fields on each item screen line:

description *keyword* *type*

The *description* characterizes the item; the *keyword* is the item identifier, and *type* specifies the kind of information associated with the item. The following item types are defined within *sysgen*:

Driver
Block/character device driver description.

Filesystem
File system handler description.

Device
Specific device or file system description.

System Devices
Specification of the logical devices that are used for necessary system functions.

Parameter
Tunable parameter specification.

Processor
Micro-processor identification declaration.

Non-Unique Driver
If enabled, this item *forces* the correct linking of a non-table driven driver, such as those for the clock, console, and MMU.

Probe
Describe how specific register locations are to be accessed and/or initialized on system initialization.

Alien Handler
Specify the presence of a non-C-language exception handler.

Multiple Handler
Define multiple handlers that are associated with a single exception.

Memory Config
Portray system memory so that the system may properly initialize and allocate memory at system startup.

CPU Board
Describe the various CPU board properties.

Include
Generate #include directive in conf.c.

EDITING

To make a collection non-deselectable you must edit the machines file of choice in **/usr/src/uts/mot/sysgen/machines** and add the line **Select fixed File** in place of **Select Description File**. This change makes all entries in the collection non-deselectable.

The default file in machines is **standard**.

COMMANDS

This section describes user commands available when *sysgen* is in interactive mode. If a command is inappropriate for a given field or screen, *sysgen* indicates an error condition by screen-flashing, bell-ringing, and/or an error message at the bottom of the screen.

All field-oriented commands apply to the *active field* displaying the → symbol. To change the active field to a different location on the screen, press the right-, left-, up-, or down-arrow key, if present or use the following keys:

h	Move left one field.
j	Move down one line.

k	Move up one line.
l	(letter `l`) Move right one field.

Other command keys include:

H	Move to the top, leftmost field.
M	Move to the center, leftmost field.
L	Move to the bottom, leftmost field.
<^y>	Scroll up 1 line.
<^e>	Scroll down 1 line.
<^u>	Scroll up half a screen.
<^d>	Scroll down half a screen.
<^f>	Move forward a screen.
<^b>	Move backward a screen.
<^l>	(letter `l`) Erase and redraw the screen.

Once the → symbol is pointing to the appropriate line or field, type **c** to change the information associated with that line or field.

c change the active field.

If this is a data field, typing **c** causes the active field to display in inverse video. To modify the data field value, type characters appropriate to the field: numeric characters in a numeric data field; numeric and alphabetic characters **a** through **f** and **A** through **F** in hexadecimal fields; any displayable characters in text fields. Use **<^h>** to delete the character underneath the cursor in a data field. Press <CR> to terminate data field entry.

s Select the active field.

If the field is part of an enable/disable list, typing **s** causes the status of the current line to be toggled; i.e., if the line was previously enabled (displaying * at the beginning of the line), the line is disabled and the * disappears or vice versa. You cannot change the status if the line is enabled and cannot be disabled (displays = at the beginning of the line).

The following keys are used to open/close *sysgen* screens:

o Open the current field to display a new screen of information associated with this field value. If *sysgen* indicates an error condition, there is no information associated with this field that can display.

q Close the current screen and return to previously opened screen.

The following command keys are appropriate for the configuration, collection, or item screen:

a Add a new entry to this screen below the line.

A Add a new entry to this screen above the line.

D Duplicate the entry indicated by the → symbol.

d Delete the entry indicated by the → symbol.

/ Search for an item name (anchored case-sensitive)

i typing **i** at any point gives a short information screen about the commands.

FILES

/usr/src/uts/mot/sysgen	*sysgen* library
/etc/sysgen	front end *sh*(1) procedure

SEE ALSO

config(1M)
master(4), dfile(4) in the *Programmer's Reference Manual.*

NAME

tic – terminfo compiler

SYNOPSIS

tic [**–v**[n]] [**–c**] *file*

DESCRIPTION

tic translates a *terminfo*(4) file from the source format into the compiled format. The results are placed in the directory **/usr/lib/terminfo**. The compiled format is necessary for use with the library routines described in *curses*(3X).

–vn
: (verbose) output to standard error trace information showing *tic*'s progress. The optional integer *n* is a number from 1 to 10, inclusive, indicating the desired level of detail of information. If *n* is omitted, the default level is 1. If *n* is specified and greater than 1, the level of detail is increased.

–c
: only check *file* for errors. Errors in **use=** links are not detected.

file
: contains one or more *terminfo*(4) terminal descriptions in source format (see *terminfo*(4)). Each description in the file describes the capabilities of a particular terminal. When a **use=***entry-name* field is discovered in a terminal entry currently being compiled, *tic* reads in the binary from **/usr/lib/terminfo** to complete the entry. (Entries created from *file* will be used first. If the environment variable **TERMINFO** is set, that directory is searched instead of **/usr/lib/terminfo**.) *tic* duplicates the capabilities in *entry-name* for the current entry, with the exception of those capabilities that explicitly are defined in the current entry.

If the environment variable **TERMINFO** is set, the compiled results are placed there instead of **/usr/lib/terminfo**.

FILES

/usr/lib/terminfo/?/*	compiled terminal description data base

SEE ALSO

curses(3X), term(4), terminfo(4) in the *Programmer's Reference Manual*.
Chapter 10 in the *Programmer's Guide*.

WARNINGS

Total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

Terminal names exceeding 14 characters will be truncated to 14 characters and a warning message will be printed.

When the **-c** option is used, duplicate terminal names will not be diagnosed; however, when **-c** is not used, they will be.

BUGS

To allow existing executables from the previous release of the operating system to continue to run with the compiled terminfo entries created by the new terminfo compiler, cancelled capabilities will not be marked as cancelled within the terminfo binary unless the entry name has a '+' within it. (Such terminal names are only used for inclusion within other entries via a **use**= entry; these names would not be used for real terminal names.)

For example:

```
4415+nl, kf1@, kf2@, ....

4415+base, kf1=\EOc, kf2=\EOd, ....

4415-nl|4415 terminal without keys,
    use=4415+nl, use=4415+base,
```

The above example works as expected; the definitions for the keys do not show up in the *4415–nl* entry. However, if the entry *4415+nl* did not have a plus sign within its name, the cancellations would not be marked within the compiled file and the definitions for the function keys would not be cancelled within *4415–nl*.

DIAGNOSTICS

Most diagnostic messages produced by *tic* during the compilation of the source file are preceded with the approximate line number and the name of the terminal currently being worked on.

mkdir ... returned bad status
: The named directory could not be created.

File does not start with terminal names in column one
: The first thing seen in the file, after comments, must be the list of terminal names.

Token after a *seek*(2) not NAMES
: Somehow the file being compiled changed during the compilation.

Not enough memory for use_list element
or
Out of memory
: Not enough free memory was available (*malloc*(3) failed).

Can't open ...
: The named file could not be created.

Error in writing ...
: The named file could not be written to.

Can't link ... to ...
: A link failed.

Error in re-reading compiled file ...
The compiled file could not be read back in.

Premature EOF
: The current entry ended prematurely.

Backspaced off beginning of line
: This error indicates something wrong happened within *tic*.

Unknown Capability - "..."
: The named invalid capability was found within the file.

Wrong type used for capability "..."
: For example, a string capability was given a numeric value.

Unknown token type
: Tokens must be followed by '@' to cancel, ',' for booleans, '#' for numbers, or '=' for strings.

"...": bad term name
or
Line ...: Illegal terminal name - "..."
Terminal names must start with a letter or digit
: The given name was invalid. Names must not contain white space or slashes, and must begin with a letter or digit.

"...": terminal name too long.
: An extremely long terminal name was found.

"...": terminal name too short.
: A one-letter name was found.

"..." filename too long, truncating to "..."
: The given name was truncated to 14 characters due to file name length limitations.

"..." defined in more than one entry. Entry being used is "...".
: An entry was found more than once.

Terminal name "..." synonym for itself
: A name was listed twice in the list of synonyms.

At least one synonym should begin with a letter.
: At least one of the names of the terminal should begin with a letter.

Illegal character - "..."
: The given invalid character was found in the input file.

Newline in middle of terminal name
: The trailing comma was probably left off of the list of names.

Missing comma
: A comma was missing.

Missing numeric value
: The number was missing after a numeric capability.

NULL string value
: The proper way to say that a string capability does not exist is to cancel it.

Very long string found. Missing comma?
: self-explanatory

Unknown option. Usage is:
: An invalid option was entered.

Too many file names. Usage is:
: self-explanatory

"..." non-existent or permission denied
: The given directory could not be written into.

"..." is not a directory
: self-explanatory

"...": Permission denied
: access denied.

"...": Not a directory
tic wanted to use the given name as a directory, but it already exists as a file

SYSTEM ERROR!! Fork failed!!!
A *fork*(2) failed.

Error in following up use-links. Either there is a loop in the links or they reference non-existent terminals. The following is a list of the entries involved:
A *terminfo*(4) entry with a **use=***name* capability either referenced a non-existent terminal called *name* or *name* somehow referred back to the given entry.

NAME

uadmin – administrative control

SYNOPSIS

/etc/uadmin *cmd fcn*

DESCRIPTION

The *uadmin* command provides control for basic administrative functions. This command is tightly coupled to the System Administration procedures and is not intended for general use. It may be invoked only by the superuser.

The arguments *cmd* (command) and *fcn* (function) are converted to integers and passed to the *uadmin* system call.

SEE ALSO

uadmin(2) in the *Programmer's Reference Manual*.

NAME

unadv – unadvertise a Remote File Sharing resource

SYNOPSIS

unadv *resource*

DESCRIPTION

unadv unadvertises a RFS *resource*, which is the advertised symbolic name of a local directory, by removing it from the advertised information on the domain name server. *unadv* prevents subsequent remote mounts of that resource. It does not affect continued access through existing remote or local mounts.

An administrator at a server can unadvertise only those resources that physically reside on the local machine. A domain administrator can unadvertise any resource in the domain from the primary name server by specifying *resource* name as *domain.resource*. (A domain administrator should only unadvertise another hosts resources to clean up the domain advertise table when that host goes down. Unadvertising another host's resource changes the domain advertise table, but not the host advertise table.)

This command is restricted to the superuser.

ERRORS

If *resource* is not found in the advertised information, an error message will be sent to standard error.

SEE ALSO

adv(1M), fumount(1M), nsquery(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* system required files and directories. Within the *uucp* makefile, it is executed before the installation takes place. It also checks for some obvious errors in the Permissions file (**/usr/lib/uucp/Permissions**). When executed with the **–v** option, it gives a detailed explanation of how the *uucp* programs will interpret the Permissions file. The **–x** option is used for debugging. *debug-level* is a single digit in the range 1-9; the higher the value, the greater the detail.

Note that *uucheck* can be used only by the superuser or *uucp*.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuscheds
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

uucico(1M), uusched(1M)
uucp(1C), uustat(1C), uux(1C) in the *User's Reference Manual*.

BUGS

The program does not check file/directory modes or some errors in the Permissions file such as duplicate login or machine name.

NAME

uucico – file transport program for the **uucp** system

SYNOPSIS

/usr/lib/uucp/uucico [**–r** *role_number*] [**–x** *debug_level*]
[**–i** *interface*] [**–d** *spool_directory*] **–s** *system_name*

DESCRIPTION

uucico is the file transport program for *uucp* work file transfers. Role numbers for the **–r** are the digit 1 for master mode or 0 for slave mode (default). The **–r** option should be specified as the digit 1 for master mode when *uucico* is started by a program or *cron*. *uux* and *uucp* both queue jobs that will be transferred by *uucico*. It is normally started by the scheduler, *uusched*, but can be started manually; this is done for debugging. For example, the shell *Uutry* starts *uucico* with debugging turned on. A single digit must be used for the **–x** option with higher numbers for more debugging.

The **–i** option defines the *interface* used with *uucico*. This interface affects only slave mode. Known interfaces are UNIX (default), TLI (basic Transport Layer Interface), and TLIS (Transport Layer Interface with STREAMS modules, read/write).

The **–d** option is used to specify the directory (**spool_directory**) that contains the work files to be transferred. The default spool directory is **/usr/spool/uucp** . The **–s** option defines the system (*system_name*) that *uucico* will try to contact. The *system_name* must be defined in the Systems file.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Devconfig
/usr/lib/uucp/Sysfiles
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), crontab(1M), uusched(1M), uutry(1M)
uucp(1C), uustat(1C), uux(1C) in the *User's Reference Manual*.

NAME

uucleanup – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uucleanup [**–C***time*] [**–W***time*] [**–X***time*] [**–m***string*] [**–o***time*] [**–s***system*] [**–D***time*] [**–W***time*] [**–X***time*] [**–m***string*] [**–o***time*] [**–s***system*] [**–x***debug level*]

DESCRIPTION

uucleanup will scan the spool directories for old files and take appropriate action to remove them in a useful way:

Inform the requester of send/receive requests for systems that cannot be reached.

Return mail that cannot be delivered to the sender.

Delete or execute *rnews* for *rnews* type files (depending on where the news originated, locally or remotely).

Remove all other files.

In addition, there is provision to warn users of requests that have been waiting for a given number of days (default 1). Note that *uucleanup* will process as if all **times** options were specified as the default values unless *time* is specifically set.

The following options are available.

–C*time*
Any **C.** files greater or equal to *time* days old will be removed with appropriate information to the requester. (default 7 days)

–D*time*
Any **D.** files greater or equal to *time* days old will be removed. An attempt will be made to deliver mail messages and execute rnews when appropriate. (default 7 days)

–W*time*
Any **C.** files equal to *time* days old will cause a mail message to be sent to the requester 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 whom to call to check the problem (**–m** option). (default 1 day)

–X*time*
Any **X.** files greater or equal to *time* days old will be removed. The **D.** files are probably not present (if they were, the **X.** could get executed). But if there are **D.** files, they will be taken care of by D. processing. (default 2 days)

–m*string*
This line will be included in the warning message generated by the **–W** option.

–o*time*
Other files whose age is more than *time* days will be deleted. (default 2 days) The default line is "See your local administrator to locate the problem".

–s*system*
Execute for *system* spool directory only.

–x*debug_level*
The **–x** debug level is a single digit between 0 and 9; higher numbers give more detailed debugging information. (If **uucleanup** was compiled with -DSMALL, no debugging output will be available.)

This program is typically started by the shell **uudemon.cln**, which should be started by *cron*(1M).

FILES

/usr/lib/uucp
directory with commands used by *uucleanup* internally

/usr/spool/uucp
spool directory

SEE ALSO

cron(1M)
uucp(1C), uux(1C) in the *User's Reference Manual.*

NAME

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

SYNOPSIS

/etc/uugetty [**–h**] [**–t** *timeout*] [**–r**] *line*
[*speed* [*type* [*linedisc*]]]
/etc/uugetty –c *file*

DESCRIPTION

uugetty is identical to *getty*(1M) but changes have been made to support using the line for *uucico, cu,* and *ct*; that is, the line can be used in both directions. The *uugetty* will allow users to login, but if the line is free, *uucico, cu,* or *ct* can use it for dialing out. The implementation depends on the fact that *uucico, cu,* and *ct* create lock files when devices are used. When the "open()" returns (or the first character is read when **–r** option is used), the status of the lock file indicates whether the line is being used by *uucico, cu, ct,* or someone trying to login. Note that in the **–r** case, several <carriage-return> characters may be required before the login message is output. The human users will be able to handle this slight inconvenience. *Uucico* trying to login will have to be told by using 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 puts out the login message, thus preventing two uugettys 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. Here is an **/etc/inittab** entry using *uugetty* on an intelligent modem or direct line:

```
30:2:respawn:/etc/uugetty –r –t 60 tty12 1200
```

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

uucico(1M), getty(1M), init(1M), tty(7)
ct(1C), cu(1C), login(1) in the *User's Reference Manual*.
ioctl(2), gettydefs(4), inittab(4) in the *Programmer's Reference Manual*.

BUGS

Ct will not work when uugetty is used with an intelligent modem such as penril or ventel.

NAME

uusched – the scheduler for the uucp file transport program

SYNOPSIS

/usr/lib/uucp/uusched [**–x** *debug_level*] [**–u** *debug_level*]

DESCRIPTION

The *uusched* program is the *uucp* file transport scheduler. It is usually started by the daemon **uudemon.hour** that is started by *cron*(1M) from an entry in **/usr/spool/cron/crontabs/uucp:**

```
39 * * * * /bin/su uucp -c "/usr/lib/uucp/uudemon.hour > /dev/null"
```

The two options are for debugging purposes only; **–x** *debug_level* will output debugging messages from *uusched* and **–u** *debug_level* will be passed as **–x** *debug_level* to *uucico*. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), crontab(1M), uucico(1M)
uucp(1C), uustat(1C), uux(1C) in the *User's Reference Manual*.

NAME

Uutry – try to contact remote system with debugging on

SYNOPSIS

/usr/lib/uucp/Uutry [**–x** *debug_level*] [**–r**] *system_name*

DESCRIPTION

Uutry is a shell that is used to invoke *uucico* to call a remote site. Debugging is turned on (default is level 5; **–x** will override that value). The **–r** overrides the retry time in **/usr/spool/uucp/.status**. The debugging output is put in file **/tmp/***system_name*. A tail **–f** of the output is executed. A <DELETE> or <BREAK> will give control back to the terminal while the *uucico* continues to run, putting its output in **/tmp/***system_name*.

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/locks/LCK*
/usr/spool/uucppublic/*
/tmp/system_name

SEE ALSO

uucico(1M).
uucp(1C), uux(1C) in the *User's Reference Manual*.

NAME

uuxqt – execute remote command requests

SYNOPSIS

/usr/lib/uucp/uuxqt [**–s** *system*] [**–x** *debug_level*]

DESCRIPTION

uuxqt is the program that executes remote job requests from remote systems generated by the use of the *uux* command. (*Mail* uses *uux* for remote mail requests). *uuxqt* searches the spool directories looking for X. files. For each X. file, *uuxqt* checks to see if all the required data files are available and accessible, and file commands are permitted for the requesting system. The **Permissions** file is used to validate file accessibility and command execution permission.

There are two environment variables that are set before the *uuxqt* command is executed:

UU_MACHINE is the machine that sent the job (the previous one).

UU_USER is the user that 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. Higher numbers give more detailed debugging information. The **-s** option refers to the system that the user is on.

FILES

/usr/lib/uucp/Permissions
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*

SEE ALSO

uucico(1M)
uucp(1C), uustat(1C), uux(1C), mail(1) in the *User's Reference Manual*.

NAME

volcopy – make literal copy of file system

SYNOPSIS

/etc/volcopy [*options*] *fsname srcdevice volname1 destdevice volname2*

DESCRIPTION

volcopy makes a literal copy of the file system using a blocksize matched to the device. *Options* are:

–a invoke a verification sequence requiring a positive operator response instead of the standard 10 second delay before the copy is made

–s (default) invoke the **DEL if wrong** verification sequence.

Other *options* are used only with tapes

–bpidensity	bits-per-inch (i.e., **800/1600/6250**)
–feetsize	size of reel in feet (i.e., **1200/2400**)
–reelnum	beginning reel number for a restarted copy
–buf	use double buffered I/O.

To use *volcopy* with media other than tape, the desired device must be linked to **/dev/rmt/device**, and that device must be specified on the volcopy command line. The following option is suggested. Specify **–bpi 1600** and use the appropriate size:

5 ¼-inch floppy	44 ft (slice 7)

The program requests length and density information if it is not given on the command line or is not recorded on an input tape label. If the file system is too large to fit on one reel, *volcopy* will prompt for additional reels. Labels of all reels are checked. Tapes may be mounted alternately on two or more drives. If *volcopy* is interrupted, it will ask if the user wants to quit or wants a shell. In the latter case, the user can perform other operations (e.g., *labelit*) and return to *volcopy* by exiting the new shell.

The *fsname* argument represents the mounted name (e.g., **root, u1**) of the filsystem being copied.

The *srcdevice* or *destdevice* should be the physical disk section or tape (e.g., **/dev/dsk/***cntlr*_**0s2**, **/dev/rdsk/***cntlr*_**1s2**).

The *volname* is the physical volume name and should match the external label sticker. Such label names are limited to six or fewer characters. *volname* may be – to use the existing volume name.

srcdevice and *volname1* are the device and volume from which the copy of the file system is being extracted. *destdevice* and *volname2* are the target device and volume.

fsname and *volname* are recorded in the last 12 characters of the superblock (**char fsname[6], volname[6];**).

When using 60Mb or 150Mb tapes use the following table to figure out the number of feet of tape to use for the internal calculations in **volcopy(1m)**.

The following table is used for 60 Mb tapes:

Bits Per Inch	**Blocks Per Foot**	**# of Blocks**	**Size of Reel (in feet)**
6250	120	120000	1000
1600	28	120000	4285 (rounded up)
800	15	120000	8000

The following table is used for 150 Mb tapes:

Bits Per Inch	**Blocks Per Foot**	**# of Blocks**	**Size of Reel (in feet)**
6250	120	300000	2500
1600	28	300000	10715 (rounded up)
800	15	300000	20000

The following formula was used for the calculations:

(size of tape in 512 byte blocks) / (Blocks Per Foot) = (Size of reel in feet)

FILES

/etc/log/filesave.log	a record of file systems/volumes copied

SEE ALSO

labelit(1M)
fs(4) in the *Programmer's Reference Manual*.
sh(1) in the *User's Reference Manual*.

WARNINGS

volcopy does not support tape-to-tape copying. Use *dd*(1M) for tape-to-tape copying.

NAME

whodo – who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

whodo produces formatted and dated output from information in the **/etc/utmp** and **/etc/ps_data** files.

The display is headed by the date, time and machine name. For each user logged in, device name, user-id and login time is shown, followed by a list of active processes associated with the user-id. The list includes the device name, process-id, cpu minutes and seconds used, and process name.

EXAMPLE

The command:

```
whodo
```

produces a display like this:

```
Tue Mar 12 15:48:03 1985
bailey

tty09   mcn     8:51
    tty09   28158   0:29 sh

tty52   bdr    15:23
    tty52   21688   0:05 sh
    tty52   22788   0:01 whodo
    tty52   22017   0:03 vi
    tty52   22549   0:01 sh

xt162   lee    10:20
    tty08    6748   0:01 layers
    xt162    6751   0:01 sh
    xt163    6761   0:05 sh
    tty08    6536   0:05 sh
```

FILES

/etc/passwd
/etc/ps_data
/etc/utmp

SEE ALSO

ps(1), who(1) in the *User's Reference Manual*.

NAME

xformtrk – convert bad track list from one format to another

SYNOPSIS

xformtrk -I *opt* **-O** *opt* [**-t** *file*] [**-d** *ddefsdir*] *devtype device*

DESCRIPTION

xformtrk converts the input data, in any of the three input formats (track; head,cylinder pairs; logical-device,block pairs), to a file containing values in either track or head,cylinder format. All input entries must be in the same format and, for any format, must be one number or pair per line. For example:

tracks:	head,cyl pair:	logical-dev,block offset pair:
1501	4 1234	11 1234
2345	6 9182	73 5678

The output from *xformtrk* goes into the file **/tmp/xf_***device* in a format suitable as input for *fblkgen*(1M). The file is removed next time the system is rebooted; it should, therefore, be moved to a more permanent place if you wish to keep it. If a file **/tmp/xf_***device* already exists, it is overwritten.

fblkgen discards those input values having corresponding track numbers which are less than zero or larger than the maximum as computed from entries in **/etc/dskdefs/***devtype*.

Values for the logical-device,block offset format normally come from the output of *errpt*(1M). The two fields of interest are:

Logical Device	<decimal #>	(<octal #>)
Block No. in Logical Filesystem	<decimal #>	
	(*in 512-byte blocks, beginning with 0*)	

Always use the decimal value of "logical device" as input to *xformtrk*.

The options are:

–I *opt*
Specify the input as either **t** for track; **c** for head,cylinder pairs; or **d** for logical-device,block pairs, one entry per line. For head,cylinder pairs, do not input the byte offset or BFI (bytes from index).

–O *opt*
Specify the output as either **t** for track or **c** for head,cylinder pairs, one entry per line.

–t *file*
Use the named file as input. The format is either single track numbers; head,cylinder pairs; or logical-device,block offset pairs, one number or pair per line. For head, cylinder pairs, do not input the byte offset or BFI (bytes from index). When the **-t** option is omitted, you are prompted to enter the data manually.

–d ddefsdir
Use the directory **ddefsdir** rather than **/etc/dskdefs/** to find the entry for *devtype.*

devtype
This is the device type found in **/etc/dskdefs** which uniquely identifies the device (e.g., m323182). See *ddefs*(1M).

device
The disk device name of the form *x_y*, where *x* is the controller and *y* is the drive logical unit number on that controller (e.g., m323_0). For device-naming conventions, see Appendix A in the *System Administrator's Guide.*

EXAMPLE

Convert the file **badtrkfile** from track numbers to head, cylinder pairs with the numbers referencing the first CDC WREN disk attached to the MVME323 controller.

```
xformtrk -I t -O c -t badtrkfile m323182 m323_0
```

SEE ALSO

fblkgen(1M)

FILES

/etc/dskdefs/*
/tmp/xf.*device*

DIAGNOSTICS

Exit Codes:

0 - success	
1 - internal failure	unknown type or device, general error
2 - I/O failure	files not found, read/write failure, etc.
3 - bad command usage	syntax error within command line
4 - user interrupt	

NAME

intro – introduction to special files

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals and SYSTEM V/88 device drivers. The names of the entries are generally derived from names for the hardware, though they are often related to the names of the special files themselves. Characteristics of both the hardware device and the corresponding SYSTEM V/88 device driver are discussed where applicable.

The naming convention creates separate subdirectories under **/dev** for each type of disk or tape device. The formats for disk devices are:

For devices other than the mvme327

/dev/{r}dsk/{r}[*cntrlr_***][***controller_number***d]***drive_numberssection_number*

For the mvme327 only

/dev/{r}dsk/{r}[*cntrlr_***][***SCSI_bus***d]***SCSI_address drive_numberssection_numbe*

where:

{ }
Fields in curly brackets represent options that affect software; they must be present if that option is being selected.

[ ]
Fields in square brackets are entirely optional; they do not affect the operation of any software or hardware; they are for informational purposes only, for the convenience of administrators, operators, and users.

r (Not Required) (The first r) indicates a raw interface to the disk. The default is normal system buffering.

dsk/
(Required) Indicates that the device is a disk.

r (Not Required) (The second r) indicates that this disk is on a remote system.

*cntrlr_*
(Not Required) Indicates the appropriate disk device in systems with multiple disk drivers. In Release 3.2 of SYSTEM V/88, the controller names are present and must be used on command lines that specify a disk device.

The disk devices available are:

m323_ ESDI Disk Controller, MVME323

m327_ SCSI Disk Controller, MVME327

*controller_number***d**
(Not Required) (For devices other than the MVME327) System administrators decide whether or not to specify the controller number in the disk device name. If the controller number is specified, the **d** introduces the drive number.

*SCSI_bus***d**
(Not Required) (For MVME327 only) Indicates the appropriate SCSI bus in systems with multiple buses. If the SCSI bus number is specified, the **d** introduces the SCSI address.

SCSI_address
(Required) (For the MVME327 only) The address of the device on the SCSI bus. The field is free format; there is no default SCSI address.

drive_number
(Required) The drive number. The field is free format; there is no default drive number. For the **mvme327**: The logical unit number of the drive at that SCSI address.

ssection_number
(Required) The section number. The field is free format; there is no default section number.

For example, the name for m327 controller 0, SCSI controller 2, drive 0, slice 2 is **/dev/dsk/m327_20s2**.

The formats for tape device names are:

For devices other than the mvme327

/dev/{r}mt/[*cntrlr_***] [***controller_number***d]** *drive_number* **[***options***] [***density***] { n }**

For mvme327 devices

/dev/{r}mt/[*cntrlr_***][***SCSI_bus***d]** *SCSI_address drive_number* **[***options***] [***density***]** { n }

where:

{ }
Fields in curly brackets represent options that affect software; they must be present if that option is being selected.

[ ]
Fields in square brackets are entirely optional; they do not affect the operation of any software or hardware; they are for informational purposes only, for the convenience of administrators, operators, and users.

r (Not Required) Indicates a raw device. The default is a blocked device.

mt/
(Required) Indicates a magnetic tape device.

*cntrlr_*
(Not Required) Indicates the appropriate magnetic tape device in systems with multiple tape drives (i.e., controller types). In Release 3.2 of SYSTEM V/88, the controller names are present and must be used on command lines that specify a tape device. The tape devices available are:

m327_ Streaming Cartridge Tape Controller, MVME327

m350_ Streaming Cartridge Tape Controller, MVME350

m355_ 9-Track Magnetic Tape Controller, MVME355

*controller_number***d**
(Not Required) (For devices other than the MVME327) System administrators decide whether or not to specify the controller number in the tape device name. If the controller number is specified, the **d** introduces the device number.

*SCSI_bus***d**
(Not Required) (For MVME327 only) Indicates the appropriate SCSI bus in systems with multiple buses. If the SCSI bus number is specified, the **d** introduces the SCSI address.

SCSI_address
(Required) (For the MVME327 only) The address of the device on the SCSI bus. The field is free format; there is no default SCSI address.

drive_number
(Required) The drive number. The field is free format; there is no default drive number. For the **mvme327**: The logical unit number of the drive at that SCSI address.

options
(Not Required) Tape options may be specified for each drive. They consist of one or more letters, described in the individual manual page for the device, which change the operating mode of the device.

density
(Not Required) Tape density may be specified for a 9-track drive. The tape density is always indicated with one of the following letters:

h (high) - 6250 bpi (or the maximum available with a particular drive)

m (medium) - 1600 bpi. This is the default when no density is specified

l (low) - 800 bpi

n (Not Required) Indicates no rewind on close. The default condition is to rewind.

As an example, the name for a 6250 bpi magnetic tape drive might be **/dev/rmt/m355_0n**. The devices often have aliases; see Appendix A of the *System Administrator's Guide.*

BUGS

While the names of the entries generally refer to vendor hardware names, in certain cases these names are seemingly arbitrary for various historical reasons.

NAME

arp – Address Resolution Protocol

SYNOPSIS

Kernel Level Interface

DESCRIPTION

ARP is a protocol used to dynamically map between DARPA Internet and physical network addresses. It is used by all *if*(7) drivers with broadcast facilities. It is not specific to Internet protocols or to Ethernet type addresses, but this implementation currently supports only that combination.

ARP caches Internet-Physical address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message which requires the mapping and broadcasts a message on the associated network requesting the address mapping. If a response is provided, the new mapping is cached and any pending message is transmitted. ARP will queue at most one packet while waiting for a mapping request to be responded to; only the most recently "transmitted" packet is kept.

To facilitate communications with systems which do not use ARP, *if*(7) *ioctl*s are provided to enter and delete entries in the Internet-to-Ethernet tables.

The interface to ARP consists of a group of kernel routines, with the requestor *if* providing the arp table, the physical network address, the internet address, the network hardware type, and the network hardware address length. Definitions for the arp table and hardware types are located in **/usr/include/netinet/arp.h**. The ARP routines can be divided into 3 areas; transmit, receive, and table management.

The routine used for transmission mapping is

mblk_t * arp_map(arptab, mp)
struct arptab *arptab;
mblk_t *mp;
A return value of NULL indicates that ARP could not resolve the internet address, but could not generate an ARP_REQUEST packet. In the normal case, **arp_map()** will return a message pointer to a resolved internet address with the *ifarg* structure filled in with the physical addresses. The mp->b_rptr must point to an *ifarg* structure followed by enough space for an *arphdr* structure.

When ARP packets have been received, the **arp_rcv** routine must be called to handle ARP_REQUEST's from other systems along with ARP_REPLY's destined for this host.

int arp_rcv(frame, arptab, flag)
char * frame;
struct arptab *arptab;
int flag;
The frame pointer must point to the beginning of the physical frame. A NULL return indicates that ARP handled the packet without anything to be sent to the network. The packet may be released at this time. A non-zero return indicates that **arp_rcv()** requires this packet be sent to the network. The packet pointed to by frame was modified and should be sent out on the network. A non-zero flag value signals ARP that the interface is setup as an arp server.

arp table management occurs from users via *ioctl*(2) requests to the *ip*(7) module using the indirect *ioctl* form. Those requests are routed to the correct *if* driver which then uses the appropriate ARP routine to apply the request to the arp table for the interface. To add an entry :

int arp_add(arptab, internet, physical)
struct arptab *arptab;
struct HOST internet;
struct e_addr *physical;
Entries added by **arp_add()** are marked as permanent, and are only removed by **arp_delete()**. All other entries are aged, and may be removed (by **arp_map()**) when *arp* table space is filled.

To remove an entry use:

int arp_delete(arptab, internet)
struct arptab *arptab;
struct HOST internet;

To clear all entries:

int arp_clear(arptab)
struct arptab *arptab;

DIAGNOSTICS

Duplicate IP address %x from %ethernet
ARP has discovered another host on the local network which responds to mapping requests for its own Internet address.

BUGS

Currently ARP will not save the packet that initiated the ARP request. This means the first packet sent to a host not found in the arp table will be lost.

Even though the different hardware types are supported, the physical address is still limited to Ethernet structure.

SEE ALSO

inet(4), if(7), arp(1M), ip(7).
"An Ethernet Address Resolution Protocol," RFC826, Dave Plummer, Network Information Center, SRI.

NAME

clone – open any minor device on a STREAMS driver

DESCRIPTION

clone is a STREAMS software driver that finds and opens an unused minor device on another STREAMS driver. The minor device passed to *clone* during the open is interpreted as the major device number of another STREAMS driver for which an unused minor device is to be obtained. Each such open results in a separate *stream* to a previously unused minor device.

The *clone* driver consists solely of an open function. This open function performs all of the necessary work so that subsequent system calls (including *close(2)*) require no further involvement of *clone*.

clone will generate an ENXIO error, without opening the device, if the minor device number provided does not correspond to a valid major device, or if the driv

CAVEATS

Multiple opens of the same minor device cannot be done through the *clone* interface. Executing *stat(2)* on the file system node for a cloned device yields a different result from executing *fstat(2)* using a file descriptor obtained from opening the node.

SEE ALSO

log(7)
STREAMS Programmer's Guide.

NAME

console – console interface

DESCRIPTION

The console provides the operator interface to the SYSTEM V/88 computer.

The file **/dev/console** is the system console, and refers to an asynchronous serial data line originating from the system board. This special file implements the features described in *termio*(7).

The file **/dev/contty** refers to a second asynchronous serial data line originating from the system board. This special file implements the features described in *termio*(7).

FILES

/dev/console
/dev/contty

SEE ALSO

termio(7)

NAME

if37x – Network Interface Driver for MVME37x Boards

SYNOPSIS

MVME37x Controller Network Interface Driver

DESCRIPTION

if37x is a STREAMS driver supporting the *if*(7) definition for use with the *Common Environment* used with the MVME147, MVME372, and MVME374. *if37x* supports two types of media access control (MAC): ethernet and ISO LLC. The Ethernet type is only supported on controllers capable of supporting an 802.3 network. The interface uses *arp*(7) for address resolution and supports the standard hardware types for Ethernet and 802.x networks. A watchdog timer system is incorporated into the interface to detect controller failures.

if37x takes stream-based messages and translates them to the appropriate *common environment* event parameter blocks (EPBs). As such, *if37x* is closely coupled to the MAC driver residing on the controller. Since *if37x* uses the controller MAC, care must be taken when using other products that use the MAC to ensure that packets are delivered to the correct recipient.

Packets received on an 802.3 network will have the same format regardless of whether the packet is 802.2 MAC or Ethernet based. The packet consists of a source address, destination address, and a size/type field. In an 802.2 network, this value is taken to be the size of the packet. In an Ethernet network, this value is used as a packet type designator. In order to route the packet to the correct recipient, the MAC checks for a valid size field of less than the maximum packet size. If the size is greater than the maximum, the size field is taken to be an Ethernet type value.

When using the Ethernet type, all packets with a size/type field of 0x800 and 0x806 will be sent to *if37x*. When using the LLC type, an LSAP of 0xaa will specify *if37x*. The LLC header is followed by a sub-network access protocol (SNAP) header which will specify the type in the same manner as with Ethernet.

Many networks will not be concerned with the use of the LLC type, since 802.3 networks and IP are almost always run as the Ethernet type. This facility is provided so that interconnection of ISO networks and Ethernet networks may be accommodated.

DIAGNOSTICS

The following message may appear in the **/usr/adm/streams** file as output of *strlog*(1M):

if37x_ip_ctl: no buffers for ctl %x
: An attempt was made to inform the upper layer of an asynchronous event, but was denied a buffer. Normal control message is to inform that the interface is down.

if37x_txllc: packet size %d
: An attempt was made to transmit a packet with a size larger than the interface MTU. The MTU for llc is 1082.

if37x_ether: packet size %d
: An attempt was made to transmit a packet over the Ethernet type which was greater than the interface MTU. The MTU for Ethernet is 1500.

if37x_pwrite: specified ip is not registered - cpu must be dead
: When attempting to write the buffer pipe to the 37x board, the *common environment* refused the interface point (ip). This is normally caused by downloading a board while the interface is active. All interfaces using this cpu will be marked as DEAD and DOWN. The stream must be dismantled and reassembled to resume normal operation.

if37x_rx: couldn't alloc buffer for ioctl_ack
: An outstanding ioctl request will be sleeping since the response buffer could not be allocated.

if37x_rx: mac send data error
: The MAC reported a transmit error.

if37x_rx: dropped pkt
: One or more packets were dropped in succession due to the inability to send the packet upstream. The message will not repeat until at least one packet has successfully been sent upstream.

if37x_rx: arp tx error
: After an arp packet was received, the response generated was denied by the *common environment*.

if37x_rx: recv pkt error
: The EPB status from the MAC indicated an error.

if37x_macstats: failed
: The request for MAC statistics was rejected by the *common environment*.

if37x_macdreg: deregister failed
: The Ethernet type deregistration failed due to a *common environment* error. The Ethernet type will still be active in the MAC.

if37x_macreg: register type overflow
: The selection of Ethernet types exceeded the maximum number allowed. The current limit is four.

if37x_macreg: register failed
: The Ethernet type registration failed due to a *common environment* error. The Ethernet type will not be active in the MAC.

if37x_ioctl: unknown control ioctl received %d
: An invalid ioctl was received on the control type.

if37xwsrv: no space for bufcall
: A data message could not be pulled up into a single message, and the bufcall request for a buffer later was denied. The entire message is discarded.

if37xclose: board error %d
: An asynchronous error was pending on the interface as a close occurred.

The following messages may be displayed on the system console :

if37xwput: unknown STR msg type %o received mp = %x
: An unsupported STREAMS message was sent to *if37x* and was discarded.

if37x_init: Attempt restart for cpu %d
: After a cpu is marked DEAD, subsequent opens will attempt to restart the interface. If this message is followed by an error message, the restart was unsuccessful.

if37x_proto: unknown proto request
: Only M_PROTO messages specified in *if*(7) are supported by *if37x*.

if37x_rxllc: no stream resource - pkt size %d dropped
: A packet arrived on the llc interface and was denied a stream buffer. The packet is discarded.

if37x_rxllc: bad snap protocol
: A packet arrived on the llc interface, and the subnetwork access protocol (SNAP) was invalid.

if37x_rxether: ifp %x pkt dropped - size %d
: A packet arrived on the Ethernet interface and was denied a stream buffer. The packet is discarded.

if37x_pwrite: cpu %d marked dead
: The *common environment* flags indicate the interface point is not registered. The cpu and all interfaces using the cpu are marked DEAD and DOWN.

BAD response from ce epb=%x buf=%x
: An EPB was received from the *common environment* of unknown type. Any processes sleeping on the interface will be awakened.

if37x_rx: reply trace packet dropped canput failed
: An arp reply trace packet was dropped due to a flow control condition up the control stream. Packet data is still sent to the network.

if37x_rx: trace packet dropped - copymsg failed
: A received packet was not traced due to a shortage of buffers to copy the packet. Packet data is still sent to the upper layer.

if37x_rx: trace packet dropped - canput failed
: A received packet was not traced due to a flow control condition on the control stream. Packet data is sent to the upper layer.

if37x_rx: rx_int but no epb
: An interrupt was indicated, but no EPBs were in the pipe.

if37x_rx: bad interface point
: An EPB was received for an unknown/unregistered interface point.

if37x_tx: trace packet dropped - failed copymsg
: A transmit packet was not traced due to a shortage of buffers to copy the packet. Packet data is still sent to the network.

if37x_tx: trace packet dropped
: A transmit packet was not traced due to a flow control condition on the control stream. Packet data is sent to the upper layer.

if37x_ioctl: Reset failed - cpu not responding - state is %x
: The attempt to reset the cpu failed due to the failure of the cpu to respond to the common environment.

interface %d_%d - possible cable open
: The MAC driver has received an error with a TDR indication. This will only occur during packet transmission. Other errors should specify the fault.

interface %d_%d - failed test collision
: At the conclusion of packet transmission, a test collision is expected from the transceiver. Normally called SQE, the collision was not detected.

interface %d_%d - excessive collision
: The MAC reported an error of excessive collisions. This error may indicate a network cable problem. This will only occur during packet transmission.

interface %d_%d - late collision detected
: The MAC reported late collision error. This error may indicate a network cable termination problem. This will only occur during packet transmission.

interface %d_%d - carrier loss detected - possible cable short - tdr = %d
: A carrier loss was detected along with a TDR value. By using the TDR value from multiple systems, it is possible to determine the direction and relative distance to the fault.

interface %d_%d - carrier loss detected - check cable to transceiver
: The MAC has reported a carrier loss without a TDR value. Check cable between transceiver and host. Also could be an indication of a blown fuse on the host interface.

interface %d_%d - not in network - no tokens heard/passed
: This error is limited to token bus networks. The tokens passed and heard did not increment during the sample period.

FILES

Device nodes for *if37x* are built in the **/dev/inet/if** and **/dev/control/if** directories. The 8-bit minor device is encoded as follows: the top four bits specify the zero based cpu number; the lower four bits specify the interface type selector. Valid types are control, inet, and raw. Device names are built in the **/dev/inet/if** directory following the *if*(7) convention of:

if - cpu - type

concatenated together. For example, the ethernet device node for the first cpu for the *if37x* driver will be **/dev/inet/if/37x_2eth.** Notice that the cpu number follows the *common environment* convention and starts with 2. Zero and one are reserved for the *common environment*.

BUGS

TDR values are not reset by the MAC.

SEE ALSO

arp(7), if(7)

NAME

ifenp – Network Interface Driver for MVME330 Boards

SYNOPSIS

MVME330 Controller Network Interface Driver

DESCRIPTION

ifenp is a STREAMS driver supporting the *if*(7) definition for use with the with the MVME330. *ifenp* supports media access control (MAC) for Ethernet. The interface uses *arp*(7) for address resolution, and supports the standard hardware type for Ethernet/802.3 networks. A watchdog timer system is incorporated into the interface to detect controller failures.

Ifenp takes stream based messages and translates them to the appropriate MVME330 enpknl rom commands. The correct version of controller firmware must be used for reliable results. The current firmware level is **ENPKNL 4.1** .

DIAGNOSTICS

The following messages may appear in the **/usr/adm/streams** file as output of *strerr*(1M):

ifenp_ip_ctl: no buffers for ctl %x
: An attempt was made to inform the upper layer as to an asynchronous event, but was denied a buffer. Normal control message is to inform that the interface is down.

ifenp_ether: packet size %d
: An attempt was make to transmit a packet over the Ethernet type which was greater than the interface MTU. The MTU for Ethernet is 1500.

ifenp_rx: couldn't alloc buffer for ioctl_ack
: An outstanding ioctl request will be sleeping since the response buffer could not be allocated.

ifenp_rx: dropped pkt
: One or more packets were dropped in succession due to the inability to send the packet upstream. The message will not repeat until at least one packet has successfully been sent upstream.

ifenp_rx: recv pkt error
: A packet was received for the IP interface type, and the interface was marked DOWN.

ifenp_ioctl: unknown control ioctl received %d
: An invalid ioctl was received on the control interface type.

ifenpwsrv: no space for bufcall
A data message could not be pulled up into a single message, and the bufcall request for a buffer later was denied. The entire message is discarded.

The following messages may be displayed on the system console :

ifenpwput: unknown STR msg type %o received mp = %x
A unsupported STREAMS message was sent to *ifenp* and was discarded.

ifenp_init: Attempt restart for cpu %d
After a cpu is marked DEAD, subsequent opens will attempt to restart the interface. If this message is followed by an error message, the restart was unsuccessful.

ifenp_proto: unknown proto request
Only M_PROTO messages specified in *if*(7) are supported by *ifenp*.

ifenp_rxether: ifp %x pkt dropped - size %d
A packet arrived on the Ethernet interface, and was denied a stream buffer. The packet is discarded.

ifenp_rx: reply trace packet dropped - canput failed
An arp reply trace packet was dropped due to a flow control condition up the control stream. Packet data is still sent to the network.

ifenp_rx: trace packet dropped - copymsg failed
A received packet was not traced due to a shortage of buffers to copy the packet. Packet data is still sent to the upper layer.

ifenp_rx: trace packet dropped - canput failed
A received packet was not traced due to a flow control condition on the control stream. Packet data is sent to the upper layer.

ifenp_tx: trace packet dropped - failed copymsg
A transmit packet was not traced due to a shortage of buffers to copy the packet. Packet data is still sent to the network.

ifenp_tx: trace packet dropped
A transmit packet was not traced due to a flow control condition on the control stream. Packet data is sent to the upper layer.

ifenp_ioctl: Reset failed - cpu not responding - state is %x
The attempt to reset the cpu failed due to the failure of the cpu to respond to the board **RESET** command.

ifenp_ioctl: cmc reset failed - cpu failed to start
: After the **GO** command was sent to the board, the host timed out waiting for the board ready flag. The interface is not usable until this is corrected.

interface 330_%d carrier loss - possible cable short - tdr = %d
: The specified interface has detected a fault. A possible short in the cable has been detected. The tdr (time domain reflectometry) value may be used, in conjunction with other systems on the same network, to aid in discovering the fault. Transceiver connections are a common cause of this type of fault. This error is only reported on transmission.

interface 330_%d carrier loss - check cable to transceiver
: Carrier has been lost between the transceiver and the controller. Packets transmitted are lost. This error is only reported on transmission.

interface 330_%d tx error - possible cable open
: The enp kernel has notified the interface that the last packet transmitted with an error. The tdr value is set, but doesn't seem useful. Experience indicates that open or unterminated network cable will generate this error.

FILES

Device nodes for *ifenp* are built in the **/dev/inet/if** and **/dev/control/if** directories. The 8-bit minor device is encoded as follows: the top 4 bits specify the zero based cpu number and the lower 4 bits specify the interface type selector. Valid types are control, inet, and raw. Internet interface device names are built in the **/dev/inet/if** directory following the *if*(7) convention of if - cpu - type concatenated together. For example, the ethernet device node for the first cpu for the *ifenp* driver will be **/dev/inet/if/330_0eth.** Cpu numbers are assigned based on board vme address. Board zero is located at address 0xdc0000, and board one responds at 0xde0000. Standard system configuration is for use of board one.

BUGS

TDR values from the board should be interpreted as a guess, as the values generated, and their stability, depend greatly on the actual fault type.

A raw interface is not supported.

SEE ALSO

arp(7), if(7), ifstat(1M)

NAME

ip – Internet IP protocol streams multiplexer module

DESCRIPTION

ip is a streams upper/lower multiplexor module. It is the network layer protocol used by the Internet protocol family.

UPPER MULTIPLEXING

Upper multiplexing is done based on Internet protocol number. These numbers are defined in RFC 790.

By convention, all *ip* devices reside in **/dev/inet/ip.** The device responds to clone open requests as well as minor device specific open requests. The first 5 minor devices are protocol specific, and can only be opened using a non-clone request.

0	Minor device zero is reserved as a general control stream. It is used to pass I/O control requests to network interface devices, and to return their response.
1	Transmission Control Protocol (MIL-1778, RFC 793).
2	User Datagram Protocol (RFC 768).
3	Internet Control Message Protocol (RFC 792).

Upper layers using the service of the multiplexer must expect to receive data messages with continuation data buffers. Data messages sent to the *ip* multiplexer are assumed to have a preceding *ifarg* and *internet header* structure.

LOWER MULTIPLEXING

There should be one or more network interface streams devices linked below the *ip* multiplexer in order for it to perform any useful functions. Multiplexing between these devices is done as necessary in accordance with the IP specifications regarding packet routing.

Data messages from the *ip* multiplexer may have continuation buffers. Data messages sent to *ip* from the *if* driver must not have continuation data buffers.

I/O CONTROLS

ip supports the following interface *ioctl* requests. All must be issued on the control stream. The values and structures used are defined in **/usr/include/netinet/ip.h** and **/usr/include/net/if.h**. All other interface specific ioctls use an **ifioctl** structure to specify the interface to *ip*.

IIOCIFLIST
: Return a list of all the interfaces currently configured into the networking system by *tpid*(1M). This ioctl is passed an array of **ifioctl** structures, and returns the information in this array. All other interface specific ioctls use an **ifioctl** structure to specify the interface to *ip*.

IIOCSETIFADDR
: Set the Internet address of the specified interface device.

IIOCGETIFADDR
: Get the Internet address of the specified interface device.

IIOCSETIFFLAG
: Set the interface flags.

IIOCGETIFFLAG
: Get the interface flags.

IIOCSETIFMTU
: Set the interface's maximum transfer unit (MTU). It is not advisable to muck with this.

IIOCGETIFMTU
: Get the interface's MTU.

IIOCSETIF
: Set the interface's address, flags and MTU.

IIOCGETIF
: Get the interface's address, flags and MTU.

IIOCIOCTL
: Pass an I/O control request down to the interface. This ioctl is passed an area of memory that has a prefix **ifioctl** structure, followed by an **ifindir** structure (defined below), possibly followed by more data.

```
/*
 * indirect ioctl structure
 */
struct ifindir {
      long cmd;    /* io control command     */
      long bytes;  /* amt of data following  */
};
```

cmd can be from a set of general interface requests or a device specific request. The general interface requests follow.

IFI_ARP_DUMP
: Return the entire ARP table. This format is defined in **/usr/include/netinet/arp.h.**

IFI_ARP_CLEAR
: Clear the entire ARP table.

IFI_ARP_ADD
: Add a permanent entry to the ARP table.

IFI_ARP_DELETE
: Delete an entry from the ARP table.

IFI_GET_HWADDR
: Return the interface's hardware address.

IFI_SET_HWADDR
: Change the interface's hardware address.

IFI_GET_STATS
: Return generalized interface statistics counters.

The following interface flags are defined. Some can only be set by the interface device.

IF_UP
: The interface is accepting packets. This can be set by a program, but an interface may reject an attempt to set it until the address has been provided, or other device specific program controlled initialization has been done.

IF_BCAST
: The interface device supports broadcast. This flag is read-only.

IF_ARP_SERVER
: Provide ARP service to the network. This flag can be set and cleared, if the interface driver supports ARP.

IF_ARP
: This interface driver supports ARP. This flag is read-only.

IF_LOOP
: This interface driver is the loop back driver. This flag is read-only.

BUGS

A raw interface to *ip* is not provided

ip neither generates nor responds to *ip* options.

Routes are static and require user intervention.

Icmp messages are tallied for statistics, but in most cases ignored. Echo is supported.

ip's notion of a broadcast address is all 1's. Incoming packets with an internet address of 0 will be rejected.

Subnet support is not yet implemented.

SEE ALSO

arp(1M), tpid(1M), netstat(1M), ifconfig(1M), tpimux(7), inet(7), if(7), icmp(7)

NAME

lo – software loopback network interface

SYNOPSIS

loopback interface device

DESCRIPTION

The *loop* interface is a software loopback mechanism that may be used for performance analysis, software testing, and/or local communication. As with other network interfaces, the loopback interface must have network addresses assigned for each address family with which it is to be used. These addresses may be set or changed with the IFI_SET_HWADDR ioctl.

DIAGNOSTICS

lo: unknown proto request.

An unrecognized protocol request was sent from IP.

lo: funny msg type X.

An unrecognized message type was sent from upstream. The value will be displayed as X.

FILES

/dev/inet/if/lo0

BUGS

Direct use of the loopback driver (i.e., raw I/O to or from **/dev/inet/if/lo0**) is discouraged since the code follows the *if*(7) interface. Due to the nature of this interface, all data must be preceded by an *IFARG* structure.

SEE ALSO

inet(4), if(7).

NAME

log – interface to STREAMS error logging and event tracing

DESCRIPTION

log is a STREAMS software device driver that provides an interface for the STREAMS error logging and event tracing processes (*strerr*(1M), *strace*(1M)). *log* presents two separate interfaces: a function call interface in the kernel through which STREAMS drivers and modules submit *log* messages; and a subset of *ioctl*(2) system calls and STREAMS messages for interaction with a user level error logger, a trace logger, or processes that need to submit their own *log* messages.

Kernel Interface

log messages are generated within the kernel by calls to the function *strlog*:

```
strlog(mid, sid, level, flags, fmt, arg1, ...)
short mid, sid;
char level;
ushort flags;
char *fmt;
unsigned arg1;
```

Required definitions are contained in **<sys/strlog.h>** and **<sys/log.h>**. *mid* is the STREAMS module id number for the module or driver submitting the *log* message. *sid* is an internal sub-id number usually used to identify a particular minor device of a driver. *level* is a tracing level that allows for selective screening out of low priority messages from the tracer. *flags* are any combination of SL_ERROR (the message is for the error logger), SL_TRACE (the message is for the tracer), SL_FATAL (advisory notification of a fatal error), and SL_NOTIFY (request that a copy of the message be mailed to the system administrator). *fmt* is a *printf(3S)* style format string, except that %s, %e, %E, %g, and %G conversion specifications are not handled. Up to NLOGARGS (currently 3) numeric or character arguments can be provided.

User Interface

log is opened via the clone interface, **/dev/log**. Each open of **/dev/log** obtains a separate *stream* to *log*. In order to receive *log* messages, a process must first notify *log* whether it is an error logger or trace logger via a STREAMS I_STR *ioctl* call (see below). For the error logger, the I_STR *ioctl* has an ic_cmd field of I_ERRLOG, with no accompanying data. For the trace logger, the *ioctl* has an ic_cmd field of I_TRCLOG, and must be accompanied by a data buffer containing an array of one or more struct trace_ids elements. Each trace_ids structure specifies an *mid*, *sid*, and *level* from which message will be accepted. *strlog* will accept messages whose *mid* and *sid* exactly match those in the trace_ids structure, and whose level is less than or equal to the level given in the trace_ids structure. A value of -1 in any of the fields of the trace_ids structure indicates that any value is accepted for that field.

At most one trace logger and one error logger can be active at a time. Once the logger process has identified itself via the *ioctl* call, *log* will begin sending up messages subject to the restrictions noted above. These messages are obtained via the *getmsg(2)* system call. The control part of this message contains a log_ctl structure, which specifies the *mid*, *sid*, *level*, *flags*, time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, and a sequence number. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of *log* messages can be determined.

Different sequence numbers are maintained for the error and trace logging *streams*, and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by NLOGARGS words for the arguments to the format string, aligned on the first word boundary following the format string.

A process may also send a message of the same structure to *log*, even if it is not an error or trace logger. The only fields of the log_ctl structure in the control part of the message that are accepted are the level and flags fields; all other fields are filled in by *log* before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to NLOGARGS) must be packed one word each, on the next word boundary following the end of the format string.

Attempting to issue an I_TRCLOG or I_ERRLOG when a logging process of the given type already exists will result in the error ENXIO being returned. Similarly, ENXIO is returned for I_TRCLOG *ioctls* without any trace_ids structures, or for any unrecognized I_STR *ioctl* calls. Incorrectly formatted *log* messages sent to the driver by a user process are silently ignored (no error results).

EXAMPLES

Example of I_ERRLOG notification.

```
struct strioctl ioc;

ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0;              /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;

ioctl(log, I_STR, &ioc);
```

Example of I_TRCLOG notification.

```
struct trace_ids tid[2];

tid[0].ti_mid = 2;
tid[0].ti_sid = 0;
tid[0].ti_level = 1;

tid[1].ti_mid = 1002;
tid[1].ti_sid = -1;       /* any sub-id will be allowed */
tid[1].ti_level = -1;     /* any level will be allowed */
```

```
        ioc.ic_cmd = I_TRCLOG;
        ioc.ic_timout = 0;
        ioc.ic_len = 2 * sizeof(struct trace_ids);
        ioc.ic_dp = (char *)tid;

        ioctl(log, I_STR, &ioc);
```

Example of submitting a *log* message (no arguments).

```
        struct strbuf ctl, dat;
        struct log_ctl lc;
        char *message = "Don't forget to pick up some milk on the way hom

        ctl.len = ctl.maxlen = sizeof(lc);
        ctl.buf = (char *)&lc;

        dat.len = dat.maxlen = strlen(message);
        dat.buf = message;

        lc.level = 0;
        lc.flags = SL_ERROR|SL_NOTIFY;

        putmsg(log, &ctl, &dat, 0);
```

FILES

/dev/log, **<sys/log.h>**, **<sys/strlog.h>**

SEE ALSO

strace(1M), strerr(1M), clone(7)
intro(2), getmsg(2), putmsg(2) in the *Programmer's Reference Manual.*
STREAMS Programmer's Guide.

NAME

lp335 - MVME335 line printer interface

DESCRIPTION

SYSTEM V/88 supports the parallel port on the system controller module with serial and parallel I/O, as well as the parallel port on the MVME335 serial and parallel I/O module as printer ports.

FILES

/dev/lp335*

SEE ALSO

lp(7), mvme335(7)
lp(1) in the *User's Reference Manual.*
ioctl(2) in the *Programmer's Reference Manual.*
MVME335 Serial and Parallel I/O Module User's Manual.

NAME

mem, kmem – core memory

DESCRIPTION

The file **/dev/mem** is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system.

Byte addresses in **/dev/mem** are interpreted as memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file **/dev/kmem** is the same as **/dev/mem** except that kernel virtual memory rather than physical memory is accessed.

FILES

/dev/mem
/dev/kmem

WARNING

Some of */dev/kmem* cannot be read because of write-only addresses or unequipped memory addresses.

BUGS

mem does not access addresses outside of physical RAM memory; hence, no driver registers are available.

NAME

mvme323 – general driver for the MVME323 ESDI Disk Controller

DESCRIPTION

The *mvme323* disk driver provides a block, physical (raw), and *ioctl*(2) interface to the MVME323 Enhanced Small Device Interface (ESDI) Controller Module.

The controller performs bad track redirection to previously assigned alternate tracks without the assistance of the driver. You must use the *dinit*(1M) program to map bad tracks to the alternate tracks.

The block device nodes **/dev/dsk/m323_*** provide access to the designated drive via the system's normal buffer cache mechanism.

The raw device nodes **/dev/rdsk/m323_*** provide for the transfer of a specified number of bytes (the byte count must be a multiple of a 512-byte physical block) between the drive and a location in the user's address space.

The driver interprets the minor device number as follows:

bit 4,5	defines one of four drives
bit 3,2,1,0	partition (slice)

Numerous *ioctl*(2) system calls are available to format a track, enable/disable programmable options, set controller and drive parameters, reset controllers and drives, read or write track headers, and display internal information.

⁞ILES

/dev/dsk/m323_*
/dev/rdsk/m323_*

EE ALSO

dinit(1M)
ioctl(2) in the *Programmer's Reference Manual.*
Appendix A in the *System Administrator's Guide* for details on device node nomenclature and default partition sizes.

NAME

mvme327 – MVME327 SCSI Controller Interface

DESCRIPTION

The MVME327 driver controls up to a total of eight SCSI devices. The types of disks currently supported are:

Description	*ddefs*(1M) File
300Mb CDC 94171 Wren IV	**m327cdcIV**
600Mb CDC 94181 Wren V	**m327cdcV**
1200Mb CDC Wren VII	**m327cdcVII**
1.2Mb IBM PC/AT Flopy	**m327pcat**
5¼" DS/DD Floppy	**m327dsdd5**

Note that each entry in the *ddefs*(1M) FILE column is the name of a file in the **/etc/dskdefs** directory that defines the characteristics of the disk. The following SCSI devices are not supported on the 327:

CDC Swift 104 Mb.
CDC Swift 172 Mb.
TEAC 155 MB data cassette.
TEAC floppy disk.
OMTI floppy disk.

The following streaming tapes types and Start/Stop tapes are currently supported:

Description	Format	Type
Archive 2150S	QIC24, QIC120, QIC150	Streamer
Exabyte EXB-8200	8MM	Streamer
Kennedy 9660	9-Track	Start/Stop

The supported disk drives handle all defects internally. There is a list of defects recorded on the medium that is automatically assigned alternates at format time. In addition, during format the drives are verified and if any additional defects are found, they are assigned alternates.

The raw device nodes **/dev/rdsk/m327_*** provide for the transfer of a specified number of bytes (byte count must be a multiple of a 512-byte physical block) between the drive and a location in the user's address space.

The MVME327 supports the following:

Scatter/gather for disk I/O
: The driver attempts to coalesce up to 64 (default) I/O requests into one I/O operation.

Multiple commands for a drive
: The driver sends each command to the controller as soon as it is received from an application (assuming the necessary system resources are available).

Simultaneous commands to multiple drives
: The driver does not have to wait for a command to complete before sending a command for another drive.

Dynamic slicing
: The slice table resides on the disk. Therefore, you do not have to configure the size and slicing of a disk into the driver. You can attach any size disk to the MVME327 without changing any configuration information. However, the following restriction applies: if the disk is to have the system booted from it, *dfile*(4) must have the proper values for the *root*, *pipe*, *swap* and *dump* device definitions.

The driver interprets the major, minor numbers as follows:

```
            MAJOR               |            MINOR
------------------------------------------------------------------
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
------------------------------------------------------------------
                  |   |       |   |       |   |   |           |
                  |   ---------   ---------   |   -------------
                  |       |           |       |         |
                  |       |           |       |         |
                  |      SCSI       drive     |       slice
                  |     Address               |
                  bus                       floppy
```

The bus bit defines one of two possible SCSI busses or one of two MVME327 boards.

The SCSI address defines the address of the device on the SCSI bus.

The drive number defines the logical unit number of the drive at that SCSI address.

The device is a floppy when the floppy bit is set.

The slice number defines one of 16 slices (0 to 15) on the disk. If the disk is partitioned into 8 slices, slice 7 is defined to cover the entire disk. If the disk is partitioned into 16 slices, slice 15 is defined to cover the entire disk.

For streaming tape devices, the slice number defines the type of action the device is to take.

Slice	Action
0	Truncate, Retension, Rewind
1	Truncate, Retension, No-Rewind
2	Append, Retension, Rewind
3	Append, Retension, No-rewind
4	Truncate, No-Retension, Rewind
5	Truncate, No-Retension, No-Rewind
6	Append, No-Retension, Rewind
7	Append, No-Retension, No-Rewind

For further information on streaming tape devices see *mvme350*(7).

For Start/Stop tape devices, the slice number defines the speed and density of the device.

Slice	Speed,Density, Rewind
0	Low speed, Low density, Rewind
1	Low speed, Low density, No-Rewind
2	High speed, Low density, Rewind
3	High speed, Low density, No-rewind
4	Low speed, High density, Rewind
5	Low speed, High density, No-Rewind
6	High speed, High density, Rewind
7	High speed, High density, No-Rewind

Naming Convention

The names of the nodes for the MVME327 tape devices are:

/dev/{r}mt/m327_{1d}*XY*[ta]{n}{r}

where:

{1d}
is the second board.

X
is the SCSI address.

Y
is the drive number.

{r}
is optional. For the streaming tape devices, **r** means retension. For the start/stop tape devices, **r** means low density.

[ta]
For the streaming tape devices, **t** means truncate and **a** means append. For the start/stop tape devices, **t** means low speed and **a** means high speed.

{n}/f1
is optional. For the streaming tape devices, **n** means no rewind. For the start/stop tape devices, **n** means no rewind.

The names of the nodes for the MVME327 disk devices are:

/dev/{r}dsk/m327_{1d}*XY*sZ

where:

{r}
is optional.

{1d}
is the second board.

X
is the SCSI address.

Y
is the drive number.

Z
is the slice number.

Local Floppy

The MVME327 supports two floppy drives that are not attached to the SCSI bus. You may access dual speed drives as "double density" or "high density" (1.2Mb) formats.

/dev/{r}dsk/m327_{1d}[ds][78]0s[07]

where:

{r}
is optional.

{1d}
is the second board.

[ds]
is either **d** for double density or **s** for high density.

[78]
is either **7** or **8**.

[07]
is either **0** for usable part of disk or **7** for entire disk.

Configuration considerations

The following is the MVME327 driver's default configuration:

SCSI Address	Device
0	Hard disk
1	Hard disk
2	Hard disk
3	Hard disk
4	Tape device
5	Tape device
6	Hard disk

To change the SCSI addresses of this configuration, change the "m327params" table in *m327space.h* and rebuild a new kernel. Refer to *sysgen*(1M).

ioctl(2) Processing

The MVME327 driver supports several *ioctl*(2) functions on the character or raw devices. These functions permit control beyond the normal *open*(2), *close*(2), *read*(2), and *write*(2) system calls.

The next two *ioctl*(2) calls have the form:

ioctl *(fildes, command, *arg)*
struct dk_ios **arg;*

The *dk_ios* structure is defined in **<sys/dk.h>**.

DKGETCFG
Gets parameters associated with the disk and stores them in the *dk_ios* structure referenced by *arg*. The MVME327 driver returns only parameters that are relevant to the MVME327 disk driver. The disk is not accessed by this command.

DKSETCFG
Sets parameters associated with the disk based on the values in the *dk_ios* structure referenced by *arg*. The MVME327 driver sets only parameters that are relevant to the MVME327 disk driver and controller. The disk is not accessed by this command.

The next two *ioctl*(2) calls have the form:

ioctl *(fildes, command, *arg)*
struct dkblk0 **arg;*

The *dkblk0* structure is defined in **<sys/dk.h>**.

DKGETINFO
Gets parameters associated with the disk and stores them in the *dkblk0* structure referenced by *arg*. The MVME327 driver returns only parameters that are relevant to the MVME327 driver and controller. There are no fields in the volume ID that are returned to the user. The disk is not accessed by this command.

DKSETINFO
Sets parameters associated with the disk based on the values in the *dkblk0* structure referenced by *arg*. The MVME327 driver sets only parameters that are relevant to the MVME327 driver and controller. The disk is not accessed by this command.

The remainder of the *ioctl*(2) calls have the form:

ioctl *(fildes, command, *arg)*
struct ioctl_struct **arg;*

The *ioctl_struct* is the structure that is referenced by *arg* in the following *ioctl*(2) descriptions:

DKFORMAT
: Formats a disk based on the information in the *dkfmt* structure referenced by *arg*. The *dkfmt* structure is defined in **<sys/dk.h>**.

DKFIXBADSPOT
: Locks out a bad spot on the disk based on the information in the *dkbadlst* structure referenced by *arg*. The *dkbadlst* structure is defined in **<sys/dk.h>**.

DKGETSLC
: Gets the slice table information for a disk and returns the information in an array of *struct slice* referenced by *arg*. The number of entries in the array is determined by the number of slices defined in the *ddefs* file. The number of slices is always a power of two; the MVME327 driver currently supports a maximum of 16 slices per Winchester disk. The disk is not accessed by this command. The slice table is defined by the *slice* structure defined in **<sys/dk.h>**.

DKSETSLC
: Sets the slice table information for a disk. Builds the slice table based on the information in an array of *struct slice* referenced by *arg*. The number of entries in the array is determined by the number of slices defined in the *ddefs* file. The number of slices is always a power of two; the MVME327 driver currently supports a maximum of 16 slices per Winchester disk. The disk is not accessed by this command. The slice table is defined by the *slice* structure defined in **<sys/dk.h>**.

The *ioctl*(2) calls, as defined in **<sys/tapectl.h>**, relevant to the tape devices are:

TAPEREWIND
: Rewinds the tape.

TAPEERASE
: Erases and then rewinds the tape.

TAPEWRTFM
: Writes a filemark onto the tape. A filemark can only be written immediately after data has been written using the *write*(2) system call.

TAPERDFM
: Reads the tape (and discards the data) up to and including the next filemark.

The *ioctl*(2) calls, as defined in **<sys/tapectl.h>**, relevant to the streaming tape devices are:

TAPERETENSION
: Retensions the tape (needed whenever a tape has not been used for some time).

TAPESETDMA
: Sets the character device DMA buffering size. The *ioctl*(2) argument parameter is the new buffer size. A buffer size of zero disables double buffering. Only the superuser may set the DMA buffer size.

TAPEGETDMA
: Returns the character device DMA (double) buffering size. The *ioctl*(2) argument parameter is the address (in the user program) of the memory location to put the DMA buffering size. If double buffering is currently enabled, this function returns the size of the smallest of the two buffers.

The *ioctl*(2) calls, as defined in **<sys/tapectl.h>**, relevant to the Start/Stop tape devices are:

TAPEHSPEED
: Set high speed on the tape drive.

TAPELSPEED
: Set low speed on the tape drive.

TAPEHDENS
: Set high density on the tape drive.

TAPELDENS
: Set low density on the tape drive.

TAPELOAD
: Load tape and set drive on line.

TAPEUNLOAD
: Set drive off line and unload tape.

dinit Considerations

dinit(1M) is the utility used to initially format the disk. It is also used to fix any new bad spots that may occur over time. The MVME327 allows redirection of new bad spots but not the saving of data. Always use the *-s* option to *dinit* when attempting to fix new bad spots.

ddefs Considerations

ddefs is the utility used to define disk characteristics. The output of the *ddefs* utility is a file that is normally saved in the **/etc/dskdefs** directory. This file is used as input to the *dinit*(1M) utility when it initializes a disk.

The fields that are important to the MVME327 are:

Comment
: Identifies the *ddefs* file to the user.

Disk type
: Decimal equivalent of a two-byte field. Upper byte is SCSI controller type; lower is peripheral type. Refer to **/usr/include/sys/mvme327.h**.

Format command
: Used by *dinit*(1M) for formatting and is set to **none** for the MVME327.

Diagnostic tracks
: Used by *dinit*(1M) to write diagnostic tracks on the disk. The default value for the MVME327 is **no**.

Bad spot strategy
: The MVME327 driver considers all media as **PERFECT**.

Maximum number of bad spots
: The maximum number of new bad spots that may be added.

Number of sectors
: The total number of sectors on the disk.

Sector size
: The physical sector size of the disk.

Sectors per track
: The number of sectors per track on the disk.

Cylinders
: The total number of cylinders on the disk.

Heads
: The number of heads on the disk.

The following fields are not used by the MVME327: *Precompensation cylinder, Sector interleave,* and *Spiral offset:*

Step rate
: Values for the step rate field for the floppy disks are in increments of 200 microseconds. Values for the step rate field for the Winchester disks are in increments of 10 microseconds. If the step rate is not what the drive expects, the MVME327 rounds up to the next nearest step rate that the drive expects.

Starting head number
: The number of the first head on the disk.

The following fields are not used by the MVME327: *ECC error length, Attributes mask, Extended attributes mask Attributes word, Gap byte 1, Gap byte 2, Gap byte 3, Gap byte 4,* and *Unformatted sector size.*

Controller attribute
: Tells the driver that the Winchester is to use its cache.

Sector slip count
: Indicates if bad spots are to be handled with sector slip. Some SCSI drives ignore this field.

The following *ddefs* utility fields have values entered based on how the disk is to be used: *slice count, root file system offset, root file system size, /usr file system size, /usr file system slice, swap size, swap slice, end-of-disk reserved area,* and *alternates list.*

tapectl(1M) Usage

You can use the *tapectl*(1M) utility to gain quick access to MVME327 tape devices. Refer to *tapectl*(1M) for details.

Error Messages

The MVME327 driver can generate several different error messages. These error messages attempt to provide enough information to permit the operator to diagnose the problem. Most error messages start with a line that prints out the drive, controller, and slice that has the error.

A misleading error message, `read error: No such device or address:` displays if you attempt to read data in "raw" mode and the "block size" specified by the command is not a multiple of 512-byte blocks.

If the error is non-recoverable (fatal), the first line of the error message for disks is:

```
FATAL ERROR on MVME327 SCSI ctl x, drive y, slice z
```

For streaming tapes it is:

```
FATAL ERROR on MVME327 SCSI ctl x, Tape drive y
```

The second line of the error message gives the error code number or a message indicating the problem.

The third line of the error message gives the logical sector number if the error occurred during a read or write operation.

The following error messages are associated with streaming tape:

Tape not ready
There may be a problem with the streaming tape cartridge. Check to see if the cartridge is in place or defective.

End of media
During write operation, ran off the end of the tape. The last file written to the tape is incomplete and needs to be written to another tape.

End of data
During read operation, tried to read past the last filemark on the tape.

Write protected
Attempted to write to a write protected tape. Remove tape cartridge from drive and check.

Other error codes may indicate serious defects. Report the error code to Motorola Field Service Division/Customer Support.

The following error messages are associated with the Winchester disk:

Drive not ready
The disk drive may not be powered up or it may have a mechanical problem. Check that the disk has power.

Drive fault
The disk drive may not be powered up or it may have a mechanical problem. Check that the disk has power.

Write protected disk
The write-protect tab is on the diskette. Remove the write-protect tab.

Unit not initialized
The drive has not been formatted.

Format failed
A format error occurred while formatting the diskette. Check the *ddefs* file and the *dinit* command line, then try to format again.

Can't fix bad spot
The attempt to fix a new bad spot failed. Try to reformat drive.

Miscellaneous Error Messages

Timeout on MVME327 SCSI ctl *x*, drive *y*, slice *z*
A request that was sent to MVME327 controller *x*, drive *y* was not returned to the driver within the allotted time. This could indicate a software or hardware problem that needs further attention.

Other error codes may indicate serious defects. Report them to Motorola Field Service Division/Customer Support.

FILES

/dev/dsk/m327_*
/dev/rdsk/m327_*
/dev/rmt/m327_*
/dev/mt/m327_*
/etc/dskdefs/m327*
/usr/include/sys/dsk.h
/usr/include/sys/dk.h
/usr/include/sys/sobpp.h
/usr/include/sys/tapectl.h
/usr/include/sys/space/m327space.h
/usr/include/sys/io/m327io.h
/usr/src/uts/mot/sysgen/descriptions/vme327

SEE ALSO

config(1M), ddefs(1M), dinit(1M), tapectl(1M), sledit(1M), mvme350(7)
open(2), close(2), ioctl(2), read(2), write(2), dfile(4), master(4) in the *Programmer's Reference Manual.*

NAME

mvme332xt – MVME332XT Communication Controller VMEmodule Interface

DESCRIPTION

The MVME332XT driver provides a general interface to the MVME332XT VMEbus Communication Controller module. The MVME332XT controller supports up to eight asynchronous serial communication ports and one Centronics compatible printer port. The MVME332XT driver supports up to eight MVME332XT controllers per system.

Each peripheral device connected to the MVME332XT has the same major device number. The MVME332XT firmware presents a generic serial and printer device interface to the driver, which can treat them as identical devices except for error message generation and interpretation. The driver distinguishes a serial device from the printer device by its device unit number. Device unit numbers 0-7 are allocated for the eight serial devices, and the printer is designated unit 8. The least significant 4 bits of the minor device field are interpreted as the device unit number. Therefore, 16 minor device numbers are required per MVME332XT controller. The four highest order bits of the minor device number are interpreted as the controller number.

read(2) Processing

The MVME332XT driver is much simpler than traditional serial I/O controller drivers since the line discipline functions are performed by the MVME332XT firmware. As such, the MVME332XT driver simply controls port modes via *ioctl(2)* calls and performs character buffer I/O to and from the MVME332XT shared RAM space via *copyout()* and *copyin()* subroutines. The MVME332XT firmware supplies the processed character data to the driver through the MVME332XT shared RAM on *read*(2) calls. The driver can then transfer the processed characters directly to the user buffer without incurring line discipline overhead.

When attempting to read from a MVME332XT device that has no data currently available:

If O_NDELAY is set from an *open*(2) or *fcntl*(2) system call, the read returns a 0.

If O_NDELAY is clear, the read blocks until data becomes available.

write(2) Processing

Writes to a MVME332XT device proceed in a similar manner. The driver transfers unprocessed character data directly to the MVME332XT shared RAM with *write*(2) calls. The MVME332XT firmware performs any character processing required to complete the character I/O.

open(2) Processing

When the *open*(2) system call is made on an MVME332XT serial device or printer, the following processing will occur:

1. If the minor device number is illegal, then the *open*(2) will fail, returning the error status **ENXIO**.
2. If **O_NDELAY** flag is set, the open returns without waiting for the carrier (serial port case) or the printer select status (printer port case). In the serial port case, carrier status is indicated by the state of the channel's DCD signal. The analogous signal for the printer port is the SELECT line, which is asserted by the printer when it has been selected.
3. If **O_NDELAY** is clear, the open blocks until the carrier (serial port case) or printer select is present. If a signal is caught while waiting for the carrier or select, the open returns with error status **EINTR**.

close(2) Processing

Upon a *close*(2) of the MVME332XT device, the minor device number must be legal, or the *close*(2) will fail, returning the error status **ENXIO**.

ioctl(2) Processing

The MVME332XT driver supports the standard *ioctl*(2) commands that apply to terminal files, and some conversion aid ioctl commands not described by *termio*(7). Several MVME332XT specific ioctl commands are also supported, which support hardware flow control and downloading of object code to the MVME332XT. Refer to the *m332xctl*(1M) manual pages and the **/usr/include/sys/mvme332xt.h** file for more details on the MVME332XT specific *ioctl* commands.

The following *ioctl*(2) system calls have the form:

```
ioctl(fildes, command, arg)
struct  termio  *arg
```

TCSETA

Set the parameters associated with the terminal from the *termio* structure referenced by **arg**. The change is immediate.

TCSETAW

Wait for the output to drain before setting *termio* parameters. The **arg** contains a pointer to the *termio* structure.

TCSETAF

Wait for the output to drain, then flush the input queue and set the *termio* parameters. The **arg** contains a pointer to the *termio* structure.

TCSETDF

Set the default *termio* parameters. The **arg** contains a pointer to the *termio* structure.

TCGETDF

Get the default *termio* parameters. The **arg** contains a pointer to the *termio* structure.

TCGETA

Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.

The following conversion *ioctl*(2) system calls have the form:

```
ioctl(fildes, command, arg)
struct  termcb  *arg
```

LDSETT

Set the parameters associated with the terminal from the termcb structure referenced by **arg**.

LDGETT

Get the parameters associated with the terminal and store in the termcb structure referenced by **arg**.

The following conversion *ioctl*(2) system calls have the form:

```
ioctl(fildes, command, arg)
struct  sgttyb  *arg
```

TIOCSETP

Set the parameters associated with the terminal from the sgttyb structure referenced by **arg**.

TIOCGETP
: Get the parameters associated with the terminal and store in the sgttyb structure referenced by **arg**.

The following *ioctl*(2) system calls have the form:

```
ioctl(fildes, command, arg)
int  arg
```

TCSBRK
: Wait for the output queue to drain. If **arg** is 0, then send a break.

TCXONC
: Start/stop control. If **arg** is 0, suspend output; if 1, restart suspended output.

TCFLSH
: If **arg** is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

The following are MVME332XT specific *ioctl* commands. The *m332xctl*(1M) utility also provides a way for the user to perform downloads and set/get hardware handshake options. The following MVME332XT specific *ioctl*(2) system calls have the form:

```
ioctl(fildes, command, arg)
struct  dl_info  *arg
```

The *dl_info* structure has the following format:

```
struct dl_info {
        unsigned long host_addr;     /* host address */
        unsigned long ipc_addr;
        unsigned long count;
        /* to be transferred */
        unsigned long                wrk0;
        unsigned short               wrk1;
};
```

The downloadable area starting address is set or returned in *dl_info.ipc_addr*, and the downloadable area size in bytes is returned in *dl_info.count*. The commands using this form are:

TCGETDL
: Get download information from the MVME332XT. The **arg** is a pointer to a *dl_info* structure. The downloadable area information is returned in the *dl_info* structure.

TCDLOAD

Download object code to the MVME332XT. The **arg** is a pointer to a *dl_info* structure described above. The *host_addr* points to the object code in the MVME332XT shared RAM. The *ipc_addr* points to the MVME332XT local RAM base address. The count is the number of bytes to be downloaded.

TCGETSYM

Get symbol table from the MVME332XT. The **arg** is a pointer to a *dl_info* structure described above. The *host_addr* points to the buffer in the MVME332XT shared RAM for the MVME332XT to return the symbol table. The *ipc_addr* should be set to 0 for the first call of the TCGETSYM to indicate the beginning of the symbol table. It is updated by the MVME332XT for subsequent TCGETSYM command. At the end of the symbol table, the MVME332XT returns EOF in the *ipc_addr* field. The count is the number of bytes returned by the MVME332XT.

TCWHAT

This command performs what function and returns SCCS id. The arg is a pointer to a *dl_info* structure described above. The *host_addr* points to the buffer in the MVME332XT shared RAM. The *ipc_addr* should be set to 0 for the first call of the TCWHAT to indicate the start of the TCWHAT function. It is updated by the MVME332XT for subsequent TCWHAT command. At the end of dumping the SCCS id, the MVME332XT returns EOF in the *ipc_addr* field. The count is the number of bytes returned by the MVME332XT.

TCLINE

Load line discipline linesw table to the MVME332XT internal data structure. The **arg** is set to point to the address of the *dl_info* structure. The *dl_info.ipc_addr* is where the linesw table starts. The *dl_info.count* is the number of lines the linesw tables contains.

The MVME332XT linesw structure is defined below:

```
struct linesw {
        int (*l_open)();
        int (*l_read)();
        int (*l_write)();
        int (*l_close)();
        int (*l_ctl)();
        int (*l_gate)();
};
```

TCEXEC

Execute a user function that is downloaded by a previous ioctl **TCDLOAD** command. The **arg** needs to be set to point to the *dl_info* structure. The *dl_info.ipc_addr* is the execution function address.

The following MVME332XT specific *ioctl*(2) system call has the form:

```
ioctl(fildes, command, arg)
int  arg
```

TCSETHW

Set hardware flow control option. If **arg** is **1**, enable hardware flow control using the RTS/CTS signal pairs; if **arg** is **0**, disable hardware flow control.

The following MVME332XT specific *ioctl*(2) system calls have the form:

```
ioctl(fildes, command, arg)
int  *arg
```

TCGETHW

Return hardware flow control status. If the specified serial port has hardware flow control enabled, **1** is returned to the **arg** integer location; otherwise, **0** is returned.

TCGETVR

Return MVME332XT firmware and driver version and revision number in the long word pointed to by **arg**. The driver version number is returned in the most significant byte, the driver revision number is in the second most significant byte, the firmware version number is in the third byte, and the firmware revision number is in the least significant byte.

TCGETDS

Get the current state of an RS-232 port or parallel printer port, such as DCD, DSR, CTS, PR_FAULT, PR_POUT, and PR_SELECT. The corresponding bits are defined in **/usr/include/sys/mvme332xt.h/f1 A one indicates the hardware pin active (or asserted).**

Error Return Codes

There are several MVME332XT specific error codes returned in u.u_error by the MVME332XT when *open*(2), *close*(2), *read*(2), *write*(2), or *ioctl*(2) are called. The following error codes are defined in **/usr/include/sys/mvme332XT.h**:

ERR_CHAN_NO: invalid channel number

The command or status channel number used in the packet is invalid.

ERR_CMD: invalid command

The command packet is invalid.

ERR_UNIT: invalid logical unit number

The device unit number is out of range; only 0 to 8 are allowed.

ERR_PARM: invalid parameter

The packet parameter is invalid.

Error messages

The MVME332XT driver generates many different error messages. These error messages, printed in English, attempt to provide information to help the operator to diagnose problems. The error messages displayed by the MVME332XT have the following format:

MVME332xt: MESSAGE on controller *X*, **unit** *Y*

where **MESSAGE** is the error message describing the symptoms, *X* is the controller number, and *Y* is the unit number.

The following are descriptions for the returned **MESSAGE**. Note that some messages include a second line, which indicates that the associated MVME332XT controller has been disabled as a result of the error condition.

Create channel error
Controller X disabled

The channel, or communication link between the driver and the MVME332XT, was not successfully created. The driver must establish the channel interface before any commands can be dispatched to the MVME332XT. This error condition typically indicates a MVME332XT configuration problem or malfunction. The controller is subsequently marked bad by the driver, and any further access attempts are disallowed.

Initialization error
Controller X disabled

An error was reported by the MVME332XT controller when the driver sent an initialization command to it. This condition will result if the driver attempts to size one of the MVME332XT read/write rings to a non base 2 value.

Unknown interrupt

An interrupt occurred from a MVME332XT controller that was marked bad or nonexistent.

Corrupted envelopes

This indicates channel corruption in the MVME332XT shared RAM.

PRINTER is deselected

This message indicates that the printer is deselected. Check the printer select switch.

PRINTER is out of paper

This indicates that the printer is out of paper. Check the printer paper supply.

PRINTER fault for unknown reason

This indicates a printer error other than the paper out or the deselected printer error conditions. Check the printer connections or refer to the printer manufacturer's user manual.

Free packet pool is empty

This error message is printed out when the driver needs to send a command to the MVME332XT device but has no free packet to use due to a MVME332XT or system level malfunction.

Device Names and Numbers

The major device number for the MVME332XT character device is 34. The minor number is assigned as follows:

Minor Number Bit Fields

MVME332XT Driver Minor Device Number Encoding

7	6	5	4	3	2	1	0
MVME332XT Controller Number				Device Unit Number			

The MVME332XT supports up to eight controllers per system. The *MVME332XT Controller Number* field is four bits wide and determines which controller is chosen. This number is counted from zero and is in the same order as the entries in **dfile.m4**. The *Device Unit Number* field is four bits wide and determines the target unit number on the selected MVME332XT controller. The following table shows the correlation between the SYSTEM V/88 device names and the devices attached to the MVME332XT controller's ports.

Special File Names

Device Name	**Minor Number**	**Controller**	**Device**
m332x00	0	0	*sp*
m332x01	1	0	*sp*
m332x02	2	0	*sp*
m332x03	3	0	*sp*
m332x04	4	0	*sp*
m332x05	5	0	*sp*
m332x06	6	0	*sp*
m332x07	7	0	*sp*
m332x08	8	0	*printer*
m332x10	16	1	*sp*
m332x11	17	1	*sp*
m332x12	18	1	*sp*
m332x13	19	1	*sp*
m332x14	20	1	*sp*
m332x15	21	1	*sp*
m332x16	22	1	*sp*
m332x17	23	1	*sp*
m332x18	24	1	*printer*
m332x*XY*	Z	X	

In the above table, *X* is the controller number, *Y* is the device number, and *Z* is the character minor number given by the following equation:

$$Z = X * 16 + Y$$

In the Device column, *sp* denotes a serial I/O port.

A new device may be installed by using the *mknod*(1M) utility with the appropriate major and minor numbers for a character type device.

SEE ALSO

mknod(1M)

NAME

mvme335 - MVME335 Serial Port Driver

DESCRIPTION

mvme335(7) driver supports the serial ports on the MVME335 Serial And Parallel I/O Module. The driver provides support for asynchronous serial communication devices. The maximum configuration consists of two MVME335 controllers, each configured with four serial ports. Each line attached to an MVME335 behaves as described in *termio*(7). The line speed of each device can be changed under software control (output speed = input speed). Baud rates 50 and 200 are not implemented. The baud rate EXTA defines the baud rate 19200.

The number of data bits (5, 6, 7, or 8), parity (even, odd, or no) and the number of stop bits(1 or 2) are also software selectable (refer to *stty*(1)).

Additional *ioctl*(2) calls are implemented to enable hardware handshake for the receiver and transmitter using the RTS/CTS lines.

The *ioctl*(2) system calls use the following form:

#include <sys/mvme331.h>
ioctl(*fildes*, *command*, *arg***)**

Three additional commands are available:

MSETHWHAND
Set hardware handshake. *arg* is a null pointer. The start/stop input control must be enabled by setting the IXOFF bit. Refer to *termio*(7).

MCLEARHWHAND
Reset hardware handshake. *arg* is a null pointer.

MGETHWHAND
Request hardware handshake status. *arg* is a pointer to an integer. If the returned value is 1, hardware handshake is enabled; if it is 0, hardware handshake is disabled.

FILES

/dev/m335_*
/dev/tty[123456789]*

SEE ALSO

portconfig(1M), lp335(7), stty(7), termio(7)
ioctl(2), tty(4) in the *Programmer's Reference Manual.*
Appendix A of the *System Administrator's Guide.*
MVME335 Serial And Parallel I/O Module User's Manual.

NAME

mvme336 – MVME336 Serial Communication Controller

DESCRIPTION

The MVME336 driver supports up to ninety-six asynchronous serial communication ports and one MVME336 controller per system. Each DeltaLink Server supports 16 ports, and the system supports 6 DeltaLink Servers.

The MVME336 subsystem supports all of the features of *termio(7).*

The minor device numbering scheme uses the letters *a* through *f* to denote the DeltaLink Server to which the port is attached. Each port is numbered *01* through *16.*

For example:

ttya01 is Server A, port 01

ttyf16 is Server F, port 16

The DeltaLink Server is *connected* to the DeltaLink Hub when the *stat* light on the Server goes from a *blinking* state to an *off* state.

The DeltaLink serial driver initially sets the port state to dead, and any *opens* return an error. When the DeltaLink Servers initialize, they send a connected message to the DeltaLink serial driver, the serial ports are marked as alive, and any subsequent opens are allowed to continue as normal.

If the link should go down (i.e., a server is powered off or the cable is disconnected) and if there is any outstanding I/O, either from the server or from the hub, the operation times out and the ports are marked as dead. Any subsequent operations returns an error. When the link is re-established, the driver is notified and a hangup signal is sent to any process using that port as a controlling tty. If there is not any I/O from the server or hub, then the link will never time out.

DEVICES

/dev/ttya*
/dev/ttyb*
/dev/ttyc*
/dev/ttyd*
/dev/ttye*
/dev/ttyf*

NAME

mvme350 – MVME350 Streamer Tape Controller VMEmodule Interface

DESCRIPTION

The MVME350 driver controls one streaming tape drive per controller. It provides advanced read/write access and tape control, having a very similar user interface to that of 9-track tapes. The general form for MVME350 file names is:

/dev/{r}mt/{bcs_}ctape**{n}**

where {r} is optional **r**, "raw" option and {n} is optional **n**, "no-rewind-on-close" option.
If the optional **r** is present, the device is the raw (or character) interface, otherwise, the device is the block interface. If the optional **n** is present, the device is the no-rewind-on-close device, otherwise the device is the rewind-on-close device. If the optional **bcs_** is present, all I/O to the tape device will be unbuffered by the MVME350 driver.

NOTE:. The way that end of media (EOM) is handled by the MVME350 driver when double buffering is not BCS compliant. Because of this, the bcs_ctape tape devices have been added. In the following discussion of double buffering, all references to the tape device refer to the non-BCS tape device.

Double Buffering

The MVME350 driver has been implemented with *double buffering* I/O processing. When making an *open*(2) system call on a character (or raw) tape device, the MVME350 will normally allocate two large system buffers to use during I/O. (See the section below regarding *open*(2) processing for exceptions.) These buffers are then used for all direct memory accesses (DMA) to and from the MVME350 Streaming Tape Controller. When making a *read*(2) system call, the data will first be read into one of the buffers, and then transferred to the reading program. When making a *write*(2) system call, the data will first be transferred from the writing program into one of the buffers. Only when the buffer is full will it be written to a streaming tape.

The advantage of this *double buffering* scheme is that most MVME350 DMA transfers will be done very efficiently, thus keeping the tape ‘‘streaming’’ and wasting little or no space on the tape due to streaming tape underruns. The default buffer size of the double buffers is 64 Kbytes for each buffer.

When reading using the double buffers, the double buffering software will attempt to read ahead of the current read point. Therefore, when the first buffer is finally exhausted, it's likely that the second buffer has already been filled. In this case, the new *read*(2) request may be completed immediately and a new read ahead may begin. The double buffering scheme will then stay ahead of the program that is reading the tape, and provide enough I/O overlap to permit efficient tape reading.

When writing using the double buffers, the double buffering software will accept new *write*(2) requests until the current buffer is full. The software will then write out the full buffer and switch to the other buffer to accept more output. If the other buffer has not yet been completely written to tape, the request will hold up the writing program until the I/O request is completed.

The default double buffer size requested by the driver may be changed by modifying the value of m350maxbsize in the *sysgen* vme350 file. Due to the requirement that the buffer be physically contiguous in memory, the system may not be able to dynamically obtain a buffer of the requested size. When this is the case, a statically allocated buffer is used instead. The size of the static buffer is determined by the value of the DBUFSZ static parameter in the *sysgen* kernel file.

open(2) Processing

When the *open*(2) system call is made on an MVME350 streaming tape, the following processing will occur:

1. If the minor device number is illegal, the *open*(2) will fail, returning the error status **ENXIO**.
2. If the tape unit is already open, the *open*(2) will fail, returning the error status **EBUSY**.
3. If the tape unit is not "on line", the *open*(2) will fail, returning the error status **EIO**.
4. If the tape unit is being opened for writing and the tape is write-protected, the *open*(2) will fail, returning the error status **EIO**.
5. If the *open*(2) call is made with the **O_NDELAY** option, no DMA buffers will be allocated and all character I/O will occur unbuffered by the MVME350 driver (Non BCS tape device only).

6. If the *open*(2) call is not made with the **O_NDELAY** option, and if the driver is unable to allocate DMA buffers on a character device open, the *open*(2) will fail, returning the error status **ENXIO** (non BCS tape device only).

NOTE: The tape is left in its current position on open.

Close(2) Processing

The following processing will occur upon a *close*(2) on a rewind-on-close tape device:

1. If the tape was opened read-only **(O_RDONLY)**, the tape is rewound to beginning of tape.
2. If the tape was opened for writing **(O_RDWR** or **O_WRONLY)**, a file mark is written at the current tape position, then the tape is rewound to beginning of tape.

The following processing will occur upon a *close*(2) on a no-rewind-on-close tape device:

1. If the tape was opened read-only **(O_RDONLY)**, and either a file mark has already been encountered or the last tape operation was a tape *ioctl*(2), then the tape is left in its current position.
2. If the tape was opened read-only **(O_RDONLY)**, a file mark has not been encountered, and the last tape operation was not a tape *ioctl*(2), then the tape is advanced to just after the next filemark.
3. If the tape was opened for writing **(O_RDWR** or **O_WRONLY)**, a file mark is written at the current tape position and the tape is positioned immediately after this file mark.

When a character (raw) device is closed, the double buffers (if any) will normally be deallocated (non BCS tape device only). The only exception to this rule occurs when an *end-of-media* is encountered while writing a streamer tape. This condition occurs when the program attempts to write off the end of a tape. The operator is notified of the condition by a message printed on the system console. The MVME350 driver will then retain the double buffers and all data currently residing in them (which is not yet written to tape).

If the same process then makes an *open*(2) request (for writing) on the same tape, all remaining data in the double buffers will be written to tape **prior** to any new I/O requests. In this way, programs like *cpio*(1) may be used to write files that span more than one tape. If any other process opens the device next, or if the device is not opened for writing, the double buffers will be flushed (with an error message printed on the system console).

In all cases, the MVME350 driver will not complete *close*(2) processing until all streaming tape operations are done. Thus, the streaming tape should not be ejected before the program that is using the streaming tape is done.

ioctl(2) Processing

The MVME350 driver supports several *ioctl*(2) functions on the character or raw device. These functions permit control functions beyond the normal *open*(2), *close*(2), *read*(2), and *write*(2) system calls. The following functions, as specified by the *ioctl*(2) request parameter, are supported:

MTIOCTOP

performs various operations on the tape as specified by the mt_op field of the mtop structure, passed in the *ioctl*(2) argument parameter - see *mtio*(4)

MTWEOF – Write end-of-file record(s) at the current tape position; the number of end-of-file records written is specified by mt_count.

MTFSF – Forward space file(s) from the current tape position; mt_count specifies the number of files to be spaced forward.

MTFSR – Forward space record(s) from the current tape position; mt_count specifies the number of records to be spaced forward.

MTREW, MTOFFL – Rewind the tape.

MTRET – Retension the tape.

MTERA – Erase the tape.

MTIOCGET

returns a status structure for the magnetic tape device. The status structure is defined in **/usr/include/sys/mtio.h** as follows:

```
struct mtget {
                    short mt_type;   /* Type of magtape device */
                    short mt_dsreg; /* Drive status register */
                    short mt_erreg; /* error register */
                    short mt_resid; /* residual */
                    daddr_t mt_fileno;/* current file number */
                    daddr_t mt_blkno;/* current block number */
}
```

Note that the returned contents of the fields **mt_dsreg** and **mt_erreg** are device dependant.

M350SETDMA (Non bcs tape device only)
: sets the character device DMA buffering size. The *ioctl*(2) argument parameter is the new buffer size. A buffer size of zero disables double buffering. Only the superuser may set the DMA buffer size.

M350GETDMA (Non bcs tape device only)
: returns the character device DMA (double) buffering size. The *ioctl*(2) argument parameter is the address (in the user program) of the memory location to put the DMA buffering size. If double buffering is currently enabled, this function returns the size of the smallest of the two buffers.

M350BYTESWAP
: enables/disables the swapping of pairs of bytes in the data being read or written. If the *ioctl*(2) argument parameter is nonzero, byte swapping is enabled; if zero, byte swapping is disabled.

mt(1) Usage

The *mt*(1) utility can be used to gain quick access to the MVME350 devices. See the *mt*(1) manual page for more details. The program supports the following functions: rewind, retension, erase, and tape positioning.

Error Messages

The MVME350 generates many different error messages. These error messages, printed in English, attempt to provide enough information to permit the operator to diagnose tape problems. Most error messages start with a line that prints out the controller and drive number that has the error. The first line of the error message looks like:

```
MVME350: Error on controller 0, drive 0
```

The second and subsequent lines of the error message describe the symptoms encountered:

Filemark detected.
The last operation encountered a filemark. When encountered, a filemark will normally terminate reading and return without an error status.

Unrecoverable data error.
Some form of unrecoverable error has occurred. The operation should be retried. If the operation continues to get this error, the tape may be damaged.

End of Media.
The tape has encountered the *end-of-media* indicator. Further reading or writing of this tape is not allowed without rewinding.

Write Protected.
The tape's write-protect switch is set to **SAFE**. Normally, the *open*(2) will fail when attempting to open a write-protected tape for write.

Drive not online.
The MVME350 does not detect an "on-line" status from the streamer tape drive. Check that all cables are properly attached. If so, retry the operation. If the problem persists, the streamer tape drive is probably damaged.

Cartridge not in place.
No streamer tape cartridge has been loaded into the selected tape drive. Check to be sure the proper special file has been used to access the tape. If this problem persists, the streamer tape drive is probably damaged.

Beginning of Media.
The *beginning-of-media* has been encountered. The tape is now rewound correctly.

No data detected.
The MVME350 has not detected any data on the tape during a read operation. A read has probably been attempted past the *end-of-data*.

No file mark encountered
An attempt to find a filemark has failed. The desired filemark appears not to be on the tape.

Not at beginning of tape.
The tape is not at the *beginning-of-tape* as expected.

Tape reset did not occur.
: After every reported error, the MVME350 attempts to reset the tape drive. If the reset fails, this error message will be printed.

Timeout.
: Some internal timeout has occurred, aborting the operation. Try the operation again. If the problem persists, try a new tape or tape drive.

Bad Unit.
: A tape drive unit failed to respond. Try the operation again. If the error persists, the cables or tape drive are probably damaged.

Bad Drive.
: A tape drive unit failed to respond. Try the operation again. If the error persists, the cables or tape drive is probably damaged.

If an error message has only the first line, retry the operation. If the error persists, the MVME350 controller, the MVME350 driver software, or the streaming tape drive may be damaged or confused. In persistent errors, resetting the machine or cycling power will sometimes clear the problem.

The MVME350 driver also produces some warning messages. These warning messages are not fatal errors but are important to the operator. The warning messages are:

DMA buffers still active.
: The last write onto the tape (using double buffering) encountered the *end-of-media*. The buffers are being retained for subsequent write operations (see the section on Double Buffering above).

DMA buffers discarded.
: The new *open*(2) request is by a different program or is not requesting to write. The retained buffers are discarded (see the section on Double Buffering above).

Initialization error.
: The MVME350 initialization sequence did not succeed. The hexadecimal value following the error message should be reported to the system administrator.

FILES

/dev/rmt/*
/usr/include/sys/mvme350.h

SEE ALSO

mt(1) in the *User's Reference Manual.*
open(2), close(2), ioctl(2) in the *Programmer's Reference Manual.*
mtio(4) *Programmer's Reference Manual.*

NAME

mvme355 – driver for the MVME355 9-Track Tape Controller

DESCRIPTION

The 9-track driver provides support for system archives and backups via 1/2-inch tape drives.

The only drive currently suuported is the Pertec 9-track drive (MCD Part No. MVME859)

The 9-track driver operates on several modes accessed via the device minor number. The default device node, **rmt/m355_0**, is used for accessing most tapes. It also sets the driver for the default conditions rewind on close, low speed, medium density, and no byte swap.

Modes:

n
: no rewind on close

s
: high speed

h
: high density

b
: byte swap

For example:

```
tar -cvfb /dev/rmt/m355_0 20 /usr
ls | cpio -oBcv >/dev/rmt/m355_0hn
```

The driver handles multiple files per tape via the no rewind device node. Only a physical (raw) interface is supported.

High density mode is drive-dependent and may be 3200- or 6250-bpi depending on the drive.

The bits of the minor device are interpreted as follows:

Bits 6, 5, 4
: Drive select (0 - 7)

Bit 3
: High speed select

Bit 2
: High density select

Bit 1
  Swap bytes select

Bit 0
  Rewind on close select

FILES

/dev/rmt/m355_*
/dev/TAPE.9TRK
/dev/rmt{0|1}*

WARNINGS

Be sure to use high quality tape (preferably 6250 quality) when running high density.
The tape device will not stream.

SEE ALSO

dd(1M), volcopy(1M), finc(1M), frec(1M), labelit(1M)
bru(1), cpio(1), tar(1) in the *User's Reference Manual*.
Appendix A of the *System Administrator's Guide*.

NAME

mvmetape – MVMETAPE Generic Streamer Tape Interface

DESCRIPTION

The MVMETAPE driver controls two streaming tape drives, **/dev/r[45][01][ant]**. It provides advanced read/write access and tape control. Though the user interface is similar to that of 9-track tapes, the generic tape interface prevents some forms of tape accesses. In particular, streaming tapes may be completely rewritten (truncated) or files may be appended at the *end-of-data*. Unlike 9-track tapes, streaming tapes may not be overwritten in the middle of a tape file (terminated by a filemark). For this reason, the MVMETAPE driver only supports two types of writing: truncating (the **O_TRUNC** *open*(2) option), and appending (the **O_APPEND** *open*(2) option). If **O_TRUNC** and **O_APPEND** are both missing from an *open*(2) request, the MVMETAPE driver will supply the appropriate one depending upon the tape's minor number.

To support the "append" and "truncate" functions, two forms of MVMETAPE special files are defined: *append* devices and *truncate* devices. When an *open*(2) occurs with neither **O_TRUNC** nor **O_APPEND** present, the file will be opened for truncation on the *truncate* device and for append on the *append* device. If the **O_TRUNC** or **O_APPEND** flags are present, it does not matter whether the *truncate* device or *append* device is used.

CAUTION: the **O_TRUNC** and **O_APPEND** options to *open*(2) **override** the *truncate* and *append* special file designations. Therefore, if a program opens a *truncate* device with the **O_APPEND** option, then data will be appended. If a program opens an *append* device with the O_TRUNC option, the tape will be rewritten. When using an unknown or untried program with the MVMETAPE, first experiment by writing small amounts of information to determine whether the program uses the **O_APPEND** or **O_TRUNC** options, or neither. Some utilities may use both options in different situations; that is, some *open*(2) calls will use **O_APPEND** while others in the same program will use **O_TRUNC**, depending upon context. The following discusses the most commonly used utilities:

Double Buffering.

The MVMETAPE driver has been implemented with *double buffering* I/O processing. When making an *open*(2) system call on a character (or raw) tape device, the MVMETAPE will normally allocate two large system buffers to use during I/O. (See the following section about *open*(2) processing for exceptions.) These buffers are then used for all direct memory accesses (DMA) to and from the MVMETAPE Streaming Tape Controller.

When making a *read*(2) system call, the data is first read into one of the buffers, then transferred to the reading program. When making a *write*(2) system call, the data is first transferred from the writing program into one of the buffers. Only when the buffer is full will it be written to a streaming tape.

The advantage of this *double buffering* scheme is that most MVMETAPE DMA transfers will be done very efficiently, thus keeping the tape "streaming" and wasting little or no space on the tape due to streaming tape underruns. The default buffer size of the double buffers is 128 Kbytes for each buffer.

When reading using the double buffers, the double buffering software will attempt to read ahead of the current read point. Therefore, when the first buffer is finally exhausted, it's likely that the second buffer has already been filled. In this case, the new *read*(2) request may be completed immediately and a new read ahead may begin. The double buffering scheme will then stay ahead of the program that is reading the tape, and provide enough I/O overlap to permit efficient tape reading.

When writing using the double buffers, the double buffering software will accept new *write*(2) requests until the current buffer is full. The software then writes out the full buffer and switches to the other buffer to accept more output. If the other buffer has not yet been completely written to tape, the request will hold up the writing program until the I/O request is completed.

The double buffering software may be permanently disabled, or the double buffering buffer sizes may be changed, using the *tapectl*(1M) control program.

The default double buffer size requested by the driver may be changed by modifying the value of m350maxbsize in the *sysgen* vme350 file. Due to the requirement that the buffer be physically contiguous in memory, the system may not be able to dynamically obtain a buffer of the requested size. When this is the case, a statically allocated buffer is used instead. The size of the static buffer is determined by the value of the DBUFSZ static parameter in the *sysgen* kernel file.

open(2) Processing.

When the *open*(2) system call is made on an MVMETAPE streaming tape, the following processing occurs:

1. If the minor device number is illegal, the *open*(2) will fail, returning the error status **ENXIO**.

2. If the tape unit is already open, the *open*(2) will fail, returning the error status **ENXIO.**
3. If the tape unit is not "on line", and if an *open*(2) call is made without the **O_NDELAY** option, the system call will fail returning the error status **ENXIO.** If an I/O request is made before the device comes "online", a device I/O error will occur.
4. If the tape unit is being opened for writing and the tape is write-protected, the *open*(2) will fail, returning the error status **ENXIO.**
5. If the *open*(2) call is made with the **O_NDELAY** option, no DMA buffers will be allocated and all character I/O will occur unbuffered by the MVMETAPE driver.
6. If the *open*(2) call is not made with the **O_NDELAY** option and the driver is unable to allocate DMA buffers on a character device open, the *open*(2) will fail, returning the error status **ENXIO.**
7. When a tape is being opened for writing, it must be opened for *rewrite* or *append*. If the open request is made with the **O_TRUNC** flag (to truncate), the tape will be rewound and the tape will be rewritten. If the open request is made with the **O_APPEND** flag (to append), the tape will be positioned following the last data written onto the tape (*end-of-data*). If the open request is not made with either **O_TRUNC** or **O_APPEND**, the device minor number will be used to determine which one to use. If the open request is made with both **O_TRUNC** and **O_APPEND**, **O_TRUNC** will override and the tape will be rewritten.
8. Retensioning is based on the minor number. Minor numbers $\geq$ 128 will not retension tape; minor numbers $<$ 128 will.

Close(2) Processing.

The following processing will occur upon a *close*(2) of a tape device:

1. If the device was opened for writing, a filemark will be written onto the tape.
2. If the special file indicated that the tape needs to be rewound, the tape will be rewound ("rewind on close").
3. If the special file indicated that the tape is not to be rewound and the device was opened for reading, the tape will be positioned following the next filemark.

When a character (raw) device is closed, the double buffers (if any) will normally be deallocated. The only exception to this rule occurs when an *end-of-media* is encountered while writing a streamer tape. This condition occurs when the program attempts to write off the end of a tape. The operator is notified of the condition by a message printed on the system console. The MVMETAPE driver will then retain the double buffers and all data currently residing in them (which is not yet written to tape). If the same process then makes an *open*(2) request (for writing) on the same tape, all remaining data in the double buffers will be written to tape prior to any new I/O requests. In this way, programs like *cpio*(1) may be used to write files that span more than one tape. If any other process opens the device next, or if the device is not opened for writing, the double buffers will be flushed (with an error message printed on the system console).

In all cases, the MVMETAPE driver will not complete *close*(2) processing until all streaming tape operations are done. Thus, the streaming tape should not be ejected before the program that is using the streaming tape is done.

ioctl(2) Processing

The MVMETAPE driver supports several *ioctl*(2) functions on the character or raw device. These functions permit control functions beyond the normal *open*(2), *close*(2), *read*(2), and *write*(2) system calls. The following functions are supported:

TAPEREWIND
: rewinds the tape.

TAPEERASE
: erases and then rewinds the tape.

TAPERETENSION
: retensions the tape (needed whenever a tape has not been used for some time).

TAPEWRTFM
: writes a filemark onto the tape. A filemark can only be written immediately after data has been written using the *write*(2) system call.

TAPERDFM
: reads the tape (and discards the data) up to and including the next filemark.

TAPESETDMA

sets the character device DMA buffering size. The *ioctl*(2) argument parameter is the new buffer size. A buffer size of zero disables double buffering. Only the superuser may set the DMA buffer size.

TAPEGETDMA

returns the character device DMA (double) buffering size. The *ioctl*(2) argument parameter is the address (in the user program) of the memory location to put the DMA buffering size. If double buffering is currently enabled, this function returns the size of the smallest of the two buffers.

TAPEBYTESWAP

enables/disables the swapping of pairs of bytes in the data being read or written. If the *ioctl*(2) argument parameter is nonzero, byte swapping is enabled; if zero, byte swapping is disabled.

Human Interfaces.

The filenames used for the MVMETAPE incorporate the drive number, whether the device is a "rewind-on-close" device, and the DEFAULT write action. The general form for MVMETAPE file names is:

/dev/{r}[45]_#[ta]{n}

where:

{r}	is optional **r**
#	is the drive number
[ta]	is one of **t** or **a**, default=**t**
{n}	is optional **n**, "rewind-on-close" option

(This form does not precisely follow the convention outlined in *intro*(7). This deviation simplifies naming for mvmetape devices because a single controller can support only one drive.)

If the optional **r** is present, the device is the raw (or character) interface. If the **t** is present, and the tape is opened for writing with neither **O_TRUNC** nor **O_APPEND** specified, the tape will be opened with **O_TRUNC**. Similarly, if the **a** is present, and the tape is opened for writing with neither **O_TRUNC** nor **O_APPEND** specified, the tape will be opened with **O_APPEND** (called the *append* device). If the **n** is present, the tape will not be rewound when the tape is closed.

The filenames to use for the MVMETAPE block and character (raw) devices are:

	Block Devices	
	Rewind on Close	**No Rewind on Close**
Truncate Device	/dev/[45]_Ct	/dev/mt/m350_Ctn
Append Device	/dev/mt/m350_Ca	/dev/mt/m350_Can

	Character (or RAW) Devices	
	Rewind on Close	**No Rewind on Close**
Truncate Device	/dev/rmt/m350_Ct	/dev/rmt/m350_Ctn
Append Device	/dev/rmt/m350_Ca	/dev/rmt/m350_Can

where:

C	is the controller number to which the tape drive is attached.

For example, to use the *dd*(1) command to copy a file system from disk **0s0** to the tape on the MVMETAPE controller zero, rewrite the tape; to rewind the tape when done, execute:

```
$ dd if=/dev/rdsk/0s0 of=/dev/rmt/m350_0t
```

This command will perform a "raw" copy of the file system on disk **0s0** to the tape. The tape driver will prohibit accidental sharing of a tape drive by only permitting one open per drive at a time.

Note that **0s0** is a generic reference to physical disk drive **#0**, partition **0**, but an actual reference to such a device will be of the following form which designates the controller, disk physical unit number, and partition number:

/dev/dsk/m320_0s0	***mvme320 controller**
/dev/dsk/m360_0s0	***mvme360 controller**

Tape Usage.

Due to the *append* or *truncate* restrictions on writing a tape, the normal tape compatible utilities need some special handling. If the tape is being accessed through I/O redirection in the shell (*sh*(1)), use the *append* device. If the tape is being opened by the utility, use the *truncate* device. When writing new utilities to use the MVMETAPE, explicit uses of **O_TRUNC** and **O_APPEND** in the *open*(2) should be made.

(Note that the *labelit*(1M) utility can be used to provide initial labels for unmounted disk or tape file systems. The **-n** option provides for inital labeling of new tapes only (this destroys previous contents). The syntax for *labelit* is:

labelit *special file_system volume*

For example:

```
labelit /dev/rmt/m350_0t usr2 r2v2.2
```

The normal tape utilities need to be used in special ways:

dd(1): The *dd*(1) program will always open the output file specified by the **of=** option, with the **O_TRUNC** flag. If no such option is specified, *dd*(1) will use its standard output. Therefore, to append to a tape using *dd*(1), redirect standard output using the shell append redirection (>>) with the *append* device; do not use the **of=** option. For example, to dump **/dev/rdsk/0s0** onto the tape followed by **/dev/rdsk/1s0**, execute:

```
$ dd if=/dev/rdsk/0s0 of=/dev/rmt/m350_0t bs=32k count=600
$ dd if=/dev/rdsk/1s0 >>/dev/rmt/m350_0a bs=32k count=600
```

cpio(1): Since *cpio*(1) uses its standard output to write tapes, simply use the shell's redirection with the *append* device. For example, to rewrite a tape using *cpio*(1):

```
$ cpio -oBv <filelist >/dev/rmt/m350_0a
```

To append to a tape using *cpio*(1):

```
$ cpio -oBv <filelist >>/dev/rmt/m350_0a
```

To restore from a tape to the current directory using *cpio*(1):

```
cpio -iBcmuldv < /dev/rmt/m350_0a
```

To back up all mounted file systems with *cpio*(1):

```
$ cd /
$ find . -print | cpio -oBvc > /dev/rmt/m350_0a
```

tar(1): To specify the output tape, *tar*(1) requires the **-f** option on the command line. If the filename supplied via the **-f** option is **-**, *tar*(1) will use its standard output. In this mode, *tar*(1) may be used in the same manner as *cpio*(1).

For example, to rewrite a tape using *tar*(1):

```
$ tar -cvf /dev/rmt/m350_0t files ...
```

or

```
$ tar -cvf - files ... >/dev/rmt/m350_0t
```

For example, to archive the files in the /etc directory (and all subdirectories) using *tar*(1):

```
$ tar -cvf /dev/rmt/m350_0t /etc ...
```

To append to a tape using *tar*(1):

```
$ tar -cvf /dev/rmt/m350_0a files ...
```

or

```
$ tar -cvf - files ... >>/dev/rmt/m350_0a
```

If the verbose *tar*(1) operation is undesireable, omit the **v** option from these examples.

To restore from a *tar*(1) tape:

```
$ tar -xvf /dev/rmt/m350_0a files ...
```

For example, to restore files in the **/usr** directory (and all subdirectories) with *tar*(1):

```
$ tar -xvf /dev/rmt/m350_0a /usr
```

To review a table of contents of a *tar*(1) tape archive without performing a transfer do the following which will list the archived filenames with permissions and modes:

```
$ tar -tvf /dev/rmt/m350_0a
```

finc(1M):

To write a tape using *finc*(1M), it is first necessary to label the tape by using *labelit*(1M). For this purpose, it is better to use the *truncate* device. To label a tape and then write the tape using *finc*(1M), execute:

```
$ labelit /dev/rmt/m350_0t fsname volname -n
$ finc -m -10 /dev/rdsk/0s0 /dev/rmt/m350_0t
```

The above example copies all files on the device **/dev/rdsk/0s0** that have been modified in the last 10 days onto the streamer tape.

frec(1M):

Since *frec*(1M) does not write tapes, it does not matter whether the *truncate* device or *append* device is used. To recover a file using *frec*, execute:

frec /dev/rmt/m350_0a *i-number:filename*

where: *i-number* identifies the file to be recovered and *filename* specifies the file where the data is to be written.

volcopy(1M):

Like *finc*(1M), *volcopy*(1M) requires the tape to have a *labelit(1M)* label at the beginning. For this purpose the *truncate* device is the best to use. To label a tape and then use *volcopy*(1M) to write it, execute:

```
$ labelit /dev/rmt/m350_0t fsname volname -n
$ volcopy fsname /dev/rdsk/0s0 volname
        /dev/rmt/m350_0t volname
```

The above example copies the file system on device **/dev/rdsk/0s0** onto streamer tape. Note that *volcopy*(1M) uses "bpi" and tape length to calculate how many tapes will be needed to make the copy. Since *volcopy* was not written for use with streaming tape, it is necessary to experiment with the values of "bpi" and tape length to adequately fill the tape. In the above example, a tape length of 1000 feet and "bpi" of 800 may have been used.

When reading tapes, either the *append* or *truncate* devices may be used. To begin reading from the second or subsequent file on a tape, the tape must be spaced to that file using the "no-rewind-on-close" device. For example, to space the tape to the beginning of the second file on a tape, execute:

```
$ dd if=/dev/rmt/m350_0tn of=/dev/null
```

or

```
$ m350ctl -f2 /dev/rmt/m350_0tn
```

When the *dd*(1) completes, the tape will be left at the beginning of the second file on the tape. If another read is attempted, it will read from the second file. If there is no second file, the MVMETAPE will generate an error.

M350ctl(1M) Usage.

The *m350ctl*(1M) utility is used to gain quick access to the MVMETAPE devices. See the *m350ctl*(1M) manual page for more details. The program supports the following functions: rewind, retension, erase, tape positioning, and DMA buffer size get and set.

Error Messages

The MVMETAPE generates many different error messages. These error messages, printed in English, attempt to provide enough information to permit the operator to diagnose tape problems. Most error messages start with a line that prints out the controller and drive number that has the error. The first line of the error message looks like:

MVMETAPE: Error on controller 0, drive 0

The second and subsequent lines of the error message describe the symptoms encountered:

Filemark detected.
The last operation encountered a filemark. When encountered, a filemark will normally terminate reading and return without an error status.

Unrecoverable data error.
Some form of unrecoverable error has occurred. The operation should be retried. If the operation continues to get this error, the tape may be damaged.

End of Media.
The tape has encountered the *end-of-media* indicator. Further reading or writing of this tape is not allowed without rewinding.

Write Protected.
The tape's write-protect switch is set to **SAFE**. Normally, the *open*(2) will fail when attempting to open a write-protected tape for write.

Drive not online.
The MVMETAPE does not detect an "on-line" status from the streamer tape drive. Check that all cables are properly attached. If so, retry the operation. If the problem persists, the streamer tape drive is probably damaged.

Cartridge not in place.
No streamer tape cartridge has been loaded into the selected tape drive. Check to be sure the proper special file has been used to access the tape. If this problem persists, the streamer tape drive is probably damaged.

Beginning of Media.
The *beginning-of-media* has been encountered. The tape is now rewound correctly.

No data detected.
The MVMETAPE has not detected any data on the tape during a read operation. A read has probably been attempted past the *end-of-data*.

No file mark encountered
An attempt to find a filemark has failed. The desired filemark appears not to be on the tape.

Not at beginning of tape.
The tape is not at the *beginning-of-tape* as expected.

Tape reset did not occur.
After every reported error, the MVMETAPE attempts to reset the tape drive. If the reset fails, this error message will be printed.

Timeout.
Some internal timeout has occurred, aborting the operation. Try the operation again. If the problem persists, try a new tape or tape drive.

Bad Unit.
A tape drive unit failed to respond. Try the operation again. If the error persists, the cables or tape drive are probably damaged.

Bad Drive.
A tape drive unit failed to respond. Try the operation again. If the error persists, the cables or tape drive is probably damaged.

If an error message has only the first line, retry the operation. If the error persists, the MVMETAPE controller, the MVMETAPE driver software, or the streaming tape drive may be damaged or confused. In persistent errors, resetting the machine or cycling power will sometimes clear the problem.

The MVMETAPE driver also produces some warning messages. These warning messages are not fatal errors but are important to the operator. The warning messages are:

DMA buffers still active.
The last write onto the tape (using double buffering) encountered the *end-of-media*. The buffers are being retained for subsequent write operations (see the section on Double Buffering above).

DMA buffers discarded.
The new *open*(2) request is by a different program or is not requesting to write. The retained buffers are discarded (see the section on Double Buffering above).

Initialization error.
The MVMETAPE initialization sequence did not succeed. The hexadecimal value following the error message should be reported to the system administrator.

FILES

/dev/r40*
/dev/r41*
/dev/rmt/m350_*
/usr/include/sys/mvmetape.h

SEE ALSO

m350ctl(1M), finc(1M), frec(1M), volcopy(1M)
cpio(1), dd(1), tar(1) in the *User's Reference Manual.*
open(2), close(2), ioctl(2) in the *Programmer's Reference Manual.*

NAME

null – the null file

DESCRIPTION

Data written on the null special file, **/dev/null**, is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

prf – operating system profiler

DESCRIPTION

The special file **/dev/prf** provides access to activity information in the operating system. Writing the file loads the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file **/dev/prf** is a pseudo-device with no associated hardware.

FILES

/dev/prf

SEE ALSO

profiler(1M)

NAME

pty – pseudo-terminal driver

SYNOPSIS

#include <sys/pty.h>

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 processes an interface identical to that described in *termio*(7). However, whereas all other devices providing the interface described in *termio*(7) have a hardware device of some sort behind them, the slave device has 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.

The following **ioctl** calls apply only to pseudo–terminals:

TIOCPKT
: Enable/disable packet mode. Packet mode is enabled by specifying (by reference) a nonzero parameter; it is disabled by specifying (by reference) a zero parameter. When applied to the master side of a pseudo–terminal, each subsequent read from the terminal 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. In the latter case, the byte is an inclusive–or of zero or more of the bits:

TIOCPKT_FLUSHREAD
: Whenever the read queue for the terminal is flushed.

TIOCPKT_FLUSHWRITE
: Whenever the write queue for the terminal is flushed.

TIOCPKT_STOP
: Whenever output to the terminal is stopped with ^S.

TIOCPKT_START
: Whenever output to the terminal is restarted.

TIOCPKT_DOSTOP
: Whenever t_stopc is ^S and t_startc is ^Q.

TIOCPKT_NOSTOP
Whenever the start and stop characters are not ^S/^Q.

This mode is used by *rlogin*(1) and *rlogind*(1M) to implement a remote–echoed, locally ^S/^Q flow-controlled remote login with proper back–flushing of output; it can be used by other similar programs.

FILES

/dev/pty[p-s][0-9a-f]	master pseudo–terminals
/dev/tty[p-s][0-9a-f]	slave pseudo–terminals

BUGS

It is not possible to explicitly send an EOF or BREAK from master to slave.

NAME

ramdisk – general ramdisk driver for SYSTEM V/88

DESCRIPTION

A *ramdisk* is a portion of memory set aside by the operating system that is accessed in exactly the same manner as a disk drive but, because it is memory, it has the advantage that there is no actual device I/O, and there are no interrupts to wait for. I/O becomes simply a transfer of data from one portion of memory to another.

The SYSTEM V/88 *ramdisk* driver views each *ramdisk* device as a collection of eight slices (partitions). Each slice has a physical address in memory of the first byte and a size in bytes. Each slice is accessed through the minor device number. The minor device number is taken as an index into the "struct size" array *ramd_sizes*[] contained in the include file **/usr/include/sys/io/ramdio.h**. *ramdisk* unit zero spans slices zero through seven, unit one spans slices 8 through 15, and so on, depending on the number of *ramdisk*s defined in *dfile* (refer to *dfile*(4)). The special files **/dev/rdsk/ramd_0s***X* and **/dev/dsk/ramd_0s***X* refer to *ramdisk* unit zero where *X* is the slice number and **dsk** or **rdsk** refer to system buffered or unbuffered I/O, respectively.

dfile must be modified before system configuration so that the memory configuration specifications do not overlap *ramdisk* slices.

ramd_sizes[] may be modified before system configuration to contain slicing information to make the *ramdisk* more manageable.

SEE ALSO

config(1M)
chmod(1) in the *User's Reference Manual*.
dfile(4) in the *Programmer's Reference Manual*.

NAME

SA – devices administered by System Administration

DESCRIPTION

The files in the directories **/dev/SA** (for block devices) and the **/dev/rSA** (for raw devices) are used by System Administration to access the devices on which it operates. For devices that support more than one partition (like disks) the **/dev/(r)SA** entry is linked to the partition that spans the entire device. Not all **/dev/(r)SA** entries are used by all System Administration commands.

FILES

/dev/SA
/dev/rSA

SEE ALSO

sysadm(1) in the *User's Reference Manual*.

NAME

streamio – STREAMS ioctl commands

SYNOPSIS

#include <stropts.h>
int ioctl (*fildes, command, arg*)
int *fildes, command;*

DESCRIPTION

STREAMS (see *intro*(2)) ioctl commands are a subset of *ioctl*(2) system calls which perform a variety of control functions on *streams.* The arguments *command* and *arg* are passed to the file designated by *fildes* and are interpreted by the *stream head.* Certain combinations of these arguments may be passed to a module or driver in the *stream.*

fildes is an open file descriptor that refers to a *stream. command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl,* they are subject to the errors described there. In addition to those errors, the call will fail with *errno* set to EINVAL, without processing a control function, if the *stream* referenced by *fildes* is linked below a multiplexor, or if *command* is not a valid value for a *stream.*

Also, as described in *ioctl,* STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the *stream head* containing an error value. This causes subsequent system calls to fail with *errno* set to this value.

COMMAND FUNCTIONS

The following *ioctl* commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH

Pushes the module whose name is pointed to by *arg* onto the top of the current *stream,* just below the *stream head.* It then calls the open routine of the newly-pushed module. On failure, *errno* is set to one of the following values:

[EINVAL]

Invalid module name.

[EFAULT]
arg points outside the allocated address space.

[ENXIO]
Open routine of new module failed.

[ENXIO]
Hangup received on *fildes*.

I_POP
Removes the module just below the *stream head* of the *stream* pointed to by *fildes*. *arg* should be 0 in an I_POP request. On failure, *errno* is set to one of the following values:

[EINVAL]
No module present in the *stream*.

[ENXIO]
Hangup received on *fildes*.

I_LOOK
Retrieves the name of the module just below the *stream head* of the *stream* pointed to by *fildes*, and places it in a null terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long. An [#include <sys/conf.h>] declaration is required. On failure, *errno* is set to one of the following values:

[EFAULT]
arg points outside the allocated address space.

[EINVAL]
No module present in *stream*.

I_FLUSH
This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

FLUSHR
Flush read queues.

FLUSHW
Flush write queues.

FLUSHRW
Flush read and write queues.

On failure, *errno* is set to one of the following values:

[ENOSR]
: Unable to allocate buffers for flush message due to insufficient STREAMS memory resources.

[EINVAL]
: Invalid *arg* value.

[ENXIO]
: Hangup received on *fildes.*

I_SETSIG
: Informs the *stream head* that the user wishes the kernel to issue the SIGPOLL signal [see *signal*(2) and *sigset*(2)] when a particular event has occurred on the *stream* associated with *fildes.* I_SETSIG supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

S_INPUT
: A non-priority message has arrived on a *stream head* read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length.

S_HIPRI
: A priority message is present on the *stream head* read queue. This is set even if the message is of zero length.

S_OUTPUT
: The write queue just below the *stream head* is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.

S_MSG
: A STREAMS signal message that contains the SIGPOLL signal has reached the front of the *stream head* read queue.

A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value S_HIPRI.

Processes that wish to receive SIGPOLL signals must explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same Stream, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further SIGPOLL signals. On failure, *errno* is set to one of the following values:

[EINVAL]
arg value is invalid or *arg* is zero and process is not registered to receive the SIGPOLL signal.

[EAGAIN]
Allocation of a data structure to store the signal request failed.

I_GETSIG
Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of I_SETSIG above. On failure, *errno* is set to one of the following values:

[EINVAL]
Process not registered to receive the SIGPOLL signal.

[EFAULT]
arg points outside the allocated address space.

I_FIND
Compares the names of all modules currently present in the *stream* to the name pointed to by *arg*, and returns 1 if the named module is present in the *stream*. It returns 0 if the named module is not present. On failure, *errno* is set to one of the following values:

[EFAULT]
arg points outside the allocated address space.

[EINVAL]
arg does not contain a valid module name.

I_PEEK
Allows a user to retrieve the information in the first message on the *stream head* read queue without taking the message off the queue. *arg* points to a *strpeek* structure which contains the following members:

```
struct strbuf     ctlbuf;
struct strbuf     databuf;
long              flags;
```

The *maxlen* field in the *ctlbuf* and *databuf strbuf* structures (see *getmsg*(2)) must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets *flags* to RS_HIPRI, I_PEEK only looks for a priority message on the *stream head* read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the *stream head* read queue, or if the RS_HIPRI flag was set in *flags* and a priority message was not present on the *stream head* read queue. It does not wait for a message to arrive. On return, *ctlbuf* specifies information in the control buffer, *databuf* specifies information in the data buffer, and *flags* contains the value 0 or RS_HIPRI. On failure, *errno* is set to the following value:

[EFAULT]
arg points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.

[EBADMSG]
Queued message to be read is not valid for I_PEEK

I_SRDOPT
Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

RNORM
Byte-stream mode, the default.

RMSGD
Message-discard mode.

RMSGN
Message-nondiscard mode.

Read modes are described in *read*(2). On failure, *errno* is set to the following value:

[EINVAL]
arg is not one of the above legal values.

I_GRDOPT
Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in *read*(2). On failure, *errno* is set to the following value:

[EFAULT]
arg points outside the allocated address space.

I_NREAD

Counts the number of data bytes in data blocks in the first message on the *stream head* read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the *stream head* read queue. For example, if zero is returned in *arg*, but the *ioctl* return value is greater than zero, this indicates that a zero-length message is next on the queue. On failure, *errno* is set to the following value:

[EFAULT]

arg points outside the allocated address space.

I_FDINSERT

Creates a message from user specified buffer(s), adds information about another *stream* and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

arg points to a *strfdinsert* structure which contains the following members:

```
struct strbuf        ctlbuf;
struct strbuf        databuf;
long                 flags;
int                  fildes;
int                  offset;
```

The *len* field in the *ctlbuf strbuf* structure (see *putmsg*(2)) must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fildes* in the *strfdinsert* structure specifies the file descriptor of the other *stream*. *offset*, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where I_FDINSERT will store a pointer. This pointer will be the address of the read queue structure of the driver for the *stream* corresponding to *fildes* in the *strfdinsert* structure. The *len* field in the *databuf strbuf* structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

flags specifies the type of message to be created. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to RS_HIPRI. For non-priority messages, I_FDINSERT will block if the *stream* write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NDELAY is set. Instead, it fails and sets *errno* to EAGAIN.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether O_NDELAY has been specified. No partial message is sent. On failure, *errno* is set to one of the following values:

[EAGAIN]
: A non-priority message was specified, the O_NDELAY flag is set, and the *stream* write queue is full due to internal flow control conditions.

[ENOSR]
: Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.

[EFAULT]
: *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.

[EINVAL]
: One of the following: *fildes* in the *strfdinsert* structure is not a valid, open *stream* file descriptor; the size of a pointer plus *offset* is greater than the *len* field for the buffer specified through *ctlptr*; *offset* does not specify a properly-aligned location in the data buffer; an undefined value is stored in *flags*.

[ENXIO]
: Hangup received on *fildes* of the *ioctl* call or *fildes* in the *strfdinsert* structure.

[ERANGE]
The *len* field for the buffer specified through *databuf* does not fall within the range specified by the maximum and minimum packet sizes of the topmost *stream* module, or the *len* field for the buffer specified through *databuf* is larger than the maximum configured size of the data part of a message, or the *len* field for the buffer specified through *ctlbuf* is larger than the maximum configured size of the control part of a message.

I_FDINSERT can also fail if an error message was received by the *stream head* of the *stream* corresponding to *fildes* in the *strfdinsert* structure. In this case, *errno* will be set to the value in the message.

I_STR
Constructs an internal STREAMS ioctl message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to send user *ioctl* requests to downstream modules and drivers. It allows information to be sent with the *ioctl*, and will return to the user any information sent upstream by the downstream recipient. I_STR blocks until the system responds with either a positive or negative acknowledgement message, or until the request "times out" after some period of time. If the request times out, it fails with *errno* set to ETIME.

At most, one I_STR can be active on a *stream*. Further I_STR calls will block until the active I_STR completes at the *stream head*. The default timeout interval for these requests is 15 seconds. The O_NDELAY [see *open*(2)] flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioctl* structure which contains the following members:

```
int     ic_cmd;         /* downstream command */
int     ic_timout;      /* ACK/NAK timeout */
int     ic_len;         /* length of data arg */
char    *ic_dp;         /* ptr to data arg */
```

ic_cmd is the internal ioctl command intended for a downstream module or driver and *ic_timout* is the number of seconds (-1 = infinite, 0 = use default, >0 = as specified) an I_STR request will wait for acknowledgement before timing out. *ic_len* is the number of bytes in the data argument and *ic_dp* is a pointer to the data argument. The *ic_len* field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the *stream* can return).

The *stream head* will convert the information pointed to by the *strioctl* structure to an internal ioctl command message and send it downstream. On failure, *errno* is set to one of the following values:

[ENOSR]
Unable to allocate buffers for the *ioctl* message due to insufficient STREAMS memory resources.

[EFAULT]
arg points, or the buffer area specified by *ic_dp* and *ic_len* (separately for data sent and data returned) is, outside the allocated address space.

[EINVAL]
ic_len is less than 0 or *ic_len* is larger than the maximum configured size of the data part of a message or *ic_timout* is less than -1.

[ENXIO]
Hangup received on *fildes.*

[ETIME]
A downstream *ioctl* timed out before acknowledgement was received.

An I_STR can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the *stream head.* In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the ioctl command sent downstream fails. For these cases, I_STR will fail with *errno* set to the value in the message.

I_SENDFD
Requests the *stream* associated with *fildes* to send a message, containing a file pointer, to the *stream head* at the other end of a *stream* pipe. The file pointer corresponds to *arg,* which must be an integer file descriptor.

I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue [see *intro*(2)] of the *stream head* at the other end of the *stream* pipe to which it is connected. On failure, *errno* is set to one of the following values:

[EAGAIN]
The sending *stream* is unable to allocate a message block to contain the file pointer.

[EAGAIN]
The read queue of the receiving *stream head* is full and cannot accept the message sent by I_SENDFD.

[EBADF]
arg is not a valid, open file descriptor.

[EINVAL]
fildes is not connected to a *stream* pipe.

[ENXIO]
Hangup received on *fildes*.

I_RECVFD
Retrieves the file descriptor associated with the message sent by an I_SENDFD *ioctl* over a *stream* pipe. *arg* is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

fd is an integer file descriptor. *uid* and *gid* are the user id and group id, respectively, of the sending *stream*.

If O_NDELAY is not set [see *open*(2)], I_RECVFD will block until a message is present at the *stream head*. If O_NDELAY is set, I_RECVFD will fail with *errno* set to EAGAIN if no message is present at the *stream head*.

If the message at the *stream head* is a message sent by an I_SENDFD, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the *strrecvfd* structure. The structure is copied into the user data buffer pointed to by *arg*. On failure, *errno* is set to one of the following values:

[EAGAIN]
A message was not present at the *stream head* read queue, and the O_NDELAY flag is set.

[EBADMSG]
The message at the *stream head* read queue was not a message containing a passed file descriptor.

[EFAULT]
arg points outside the allocated address space.

[EMFILE]
NOFILES file descriptors are currently open.

[ENXIO]
Hangup received on *fildes*.

The following two commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK
Connects two *streams*, where *fildes* is the file descriptor of the *stream* connected to the multiplexing driver, and *arg* is the file descriptor of the *stream* connected to another driver. The *stream* designated by *arg* gets connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgement message to the *stream head* regarding the linking operation. This call returns a multiplexor ID number (an identifier used to disconnect the multiplexor, see I_UNLINK) on success, and a -1 on failure. On failure, *errno* is set to one of the following values:

[ENXIO]
Hangup received on *fildes*.

[ETIME]
Time out before acknowledgement message was received at *stream head*.

[EAGAIN]
Temporarily unable to allocate storage to perform the I_LINK.

[ENOSR]
Unable to allocate storage to perform the I_LINK due to insufficient STREAMS memory resources.

[EBADF]
arg is not a valid, open file descriptor.

[EINVAL]
fildes stream does not support multiplexing.

[EINVAL]
arg is not a *stream*, or is already linked under a multiplexor.

[EINVAL]
The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given *stream head* is linked into a multiplexing configuration in more than one place.

An I_LINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received a In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_LINK will fail with *errno* set to the value in the message.

I_UNLINK
Disconnects the two *streams* specified by *fildes* and *arg*. *fildes* is the file descriptor of the *stream* connected to the multiplexing driver. *fildes* must correspond to the *stream* on which the *ioctl* I_LINK command was issued to link the *stream* below the multiplexing driver. *arg* is the multiplexor ID number that was returned by the I_LINK. If *arg* is -1, then all Streams which were linked to *fildes* are disconnected. As in I_LINK, this command requires the multiplexing driver to acknowledge the unlink. On failure, *errno* is set to one of the following values:

[ENXIO]
Hangup received on *fildes*.

[ETIME]
Time out before acknowledgement message was received at *stream head*.

[ENOSR]
Unable to allocate storage to perform the I_UNLINK due to insufficient STREAMS memory resources.

[EINVAL]
arg is an invalid multiplexor ID number or *fildes* is not the *stream* on which the _LINK that returned *arg* was performed.

An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_UNLINK fails with *errno* set to the value in the message.

SEE ALSO

close(2), fcntl(2), intro(2), ioctl(2), open(2), read(2), getmsg(2), poll(2), putmsg(2), signal(2), sigset(2), write(2) in the *Programmer's Reference Manual*.
STREAMS Programmer's Guide.
STREAMS Primer.

DIAGNOSTICS

Unless specified otherwise above, the return value from *ioctl* is 0 upon success and -1 upon failure with *errno* set as indicated.

NAME

sxt – pseudo-device driver

DESCRIPTION

The special file **/dev/sxt** is a pseudo-device driver that interposes a discipline between the standard *tty* line disciplines and a real device driver. The standard disciplines manipulate *virtual tty* structures (channels) declared by the **/dev/sxt** driver. **/Dev/sxt** acts as a discipline manipulating a *real tty* structure declared by a real device driver. The **/dev/sxt** driver is currently only used by the *shl*(1) command.

Virtual ttys are named by inodes in the subdirectory **/dev/sxt** and are allocated in groups of up to eight. To allocate a group, a program should exclusively open a file with a name of the form **/dev/sxt/??0** (channel 0) and then execute a SXTIOCLINK *ioctl* call to initiate the multiplexing.

Only one channel, the *controlling* channel, can receive input from the keyboard at a time; others attempting to read will be blocked.

There are two groups of *ioctl*(2) commands supported by *sxt*. The first group contains the standard *ioctl* commands described in *termio*(7), with the addition of the following:

TIOCEXCL
: Set *exclusive use* mode: no further opens are permitted until the file has been closed.

TIOCNXCL
: Reset *exclusive use* mode: further opens are once again permitted.

The second group are commands to *sxt* itself. Some of these may only be executed on channel 0.

SXTIOCLINK
: Allocate a channel group and multiplex the virtual ttys onto the real tty. The argument is the number of channels to allocate. This command may only be executed on channel 0. Possible errors include:

  EINVAL
  : The argument is out of range.

  ENOTTY
  : The command was not issued from a real tty.

  ENXIO
  : *linesw* is not configured with *sxt*.

EBUSY
An SXTIOCLINK command has already been issued for this real *tty*.

ENOMEM
There is no system memory available for allocating the virtual tty structures.

EBADF
Channel 0 was not opened before this call.

SXTIOCSWTCH
Set the controlling channel. Possible errors include:

EINVAL
An invalid channel number was given.

EPERM
The command was not executed from channel 0.

SXTIOCWF
Cause a channel to wait until it is the controlling channel. This command will return the error, *EINVAL,* if an invalid channel number is given.

SXTIOCUBLK
Turn off the **loblk** control flag in the virtual tty of the indicated channel. The error *EINVAL* will be returned if an invalid number or channel 0 is given.

SXTIOCSTAT
Get the status (blocked on input or output) of each channel and store in the *sxtblock* structure referenced by the argument. The error *EFAULT* will be returned if the structure cannot be written.

SXTIOCTRACE
Enable tracing. Tracing information is written to the console on the 3B2 Computer. This command has no effect if tracing is not configured.

SXTIOCNOTRACE
Disable tracing. This command has no effect if tracing is not configured.

FILES

/dev/sxt/??[0-7]	Virtual TTY devices

SEE ALSO

termio(7)
shl(1), stty(1) in the *User's Reference Manual.*
ioctl(2), open(2) in the *Programmer's Reference Manual.*

NAME

tcp – Internet Transmission Control Protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

int socket(AF_INET, SOCK_STREAM, 0);

or

#include <tiuser.h>
#include <sys/tpiaddr.h>
#include <netinet/in.h>

int t_open(path, oflag, info)

DESCRIPTION

The TCP protocol provides reliable, flow-controlled, two-way transmission of data. TCP uses the standard Internet address format and, in addition, provides a per-host collection of port addresses. Thus, each address is composed of an Internet address specifying the host and network, with a specific TCP port on the host identifying the peer entity.

TCP is one of the protocols supported by *tpimux*(4) . As such, TCP has two user level interface points: sockets and the transport library interface (TLI). The socket semantics follow the BSD 4.2 implementation, while the TLI interface is defined in the *SYSTEM V/68 Programmer's Guide*. Socket address formats follow the BSD model of the **sockaddr** structure. TLI addresses use the **tpiaddr** structure. The **tpiaddr** structure represents the same address form as the **sockaddr** structure except the TLI address does not contain the zero pad bytes. Header files, for both sockets and TLI, are supplied for address manipulation. Refer to *inet*(3N), *gethostent*(3N) and *getnetent*(3N) for methods of generating addresses suitable for use in the **sockaddr** or **tpiaddr** structures.

Sockets utilizing the tcp protocol are either active or passive. Active sockets initiate connections to passive sockets. By default TCP sockets are created active; to create a passive socket the *listen*(2) system call must be used after binding the socket with the *bind*(2) system call. Only passive sockets may use the *accept*(2) call to accept incoming connections. Only active sockets may use the *connect*(2) call to initiate connections.

Passive sockets may underspecify their location to match incoming connection requests from multiple networks. This technique, termed wildcard addressing, allows a single server to provide service to clients on multiple networks. To create a socket which listens on all networks, the Internet address INADDR_ANY must be bound. The TCP port may still be specified at this time; if the port is not specified the system will assign one. Once a connection has been established the socket's address is fixed by the peer entity's location. The address assigned the socket is the address associated with the network interface through which packets are being transmitted and received. Normally this address corresponds to the peer entity's network.

The default condition for sockets is for TCP Keepalive to **not** be set. This may be changed with *setsockopt*(2) if desired. (Refer to *setsockopt*(2) for details and restrictions.)

When using TLI, the path specified in the *t_open*(3N) call will select the protocol desired and allocate a transport endpoint. For TCP service, use the device **/dev/tpimux/tcp**. The *t_bind*(3N) function associates a protocol address with the supplied transport endpoint. Since TCP is a connection oriented protocol, the *qlen* field of the bind request/return structure is used to initialize a listening protocol address. At the completion of the *t_bind*(3N) with a non-zero *qlen*, TCP will be ready to accept connections for the bound protocol address. The *t_listen*(3N) function is then used to retrieve the incoming connections. A *t_bind*(3N) with a *qlen* equal to zero will allocate a local protocol address but will not begin accepting connections. A *t_connect(3N)* is then used to initiate connections to other transport endpoints. Wildcard addressing is allowed in the same manner used for sockets.

Due to the asynchronous nature of TLI, TCP Keepalive is set **on** as the default option on TCP connections. This will generate the appropriate disconnect indications on a broken transport endpoint without the necessity of sending data.

TCP does not support the optional TLI orderly disconnect function.

DEVICES

To access TCP for use with TLI, the device **/dev/tpimux/tcp** should be used. Devices are not used when allocating a socket.

BUGS

Keepalive cannot be disabled on TLI connections.

SEE ALSO

socket(2), accept(2), listen(2), connect(2), intro(3), t_connect(3N), t_listen(3N), t_open(3N), t_bind(3N), tpimux(7), inet(4), ip(7), if(7).

NAME

termio – general terminal interface

DESCRIPTION

All of the asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open terminal files; they are opened by *getty* and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(2). A process can break this association by changing its process group using *setpgrp*(2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, the buffer is flushed and all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. 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 may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, the character # erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character @ kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The erase and kill characters may be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR
: (Rubout or ASCII DEL) generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).

QUIT
: (Control-| or ASCII FS) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.

SWTCH
: (Control-z or ASCII SUB) is used by the job control facility, *shl,* to change the current layer to the control layer.

ERASE
: (#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL
: (@) deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF
: (Control-d or ASCII EOT) may 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, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.

NL
: (ASCII LF) is the normal line delimiter. It can not be changed or escaped.

EOL
: (ASCII NUL) is an additional line delimiter, like NL. It is not normally used.

EOL2
: is another additional line delimiter.

STOP
: (Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START
: (Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, SWTCH, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a *hang-up* signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hang-up signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Several *ioctl*(2) system calls apply to terminal files. The primary calls use the following structure, defined in **<termio.h>**:

```
#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 */
};
```

The special control characters are defined by the array *c_cc*. The relative positions and initial values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	reserved	
7	SWTCH	

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map upper-case to lower-case on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, i.e., not put on the input queue, therefore, not read by any process. Otherwise, if BRKINT is set, the break condition generates an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

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). Otherwise, if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character, will restart output which has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is all-bits-clear.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.

OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

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 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. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also 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. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters will be 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, the carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2, four fill characters.

Horizontal-tab delay type 1 is dependent 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 will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud

B9600	0000015	9600 baud
B19200	0000016	19200 baud
EXTA	0000016	External A
B38400	0000017	38400 baud
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.
RCV1EN	0010000	
XMT1EN	0020000	
LOBLK	0040000	Block layer output.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

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, two stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise, no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates, i.e., the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise, modem control is assumed.

If LOBLK is set, the output of a job control layer will be blocked when it is not the current layer. Otherwise, the output generated by that layer will be multiplexed onto the current layer.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR, SWTCH, and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g., 0377).

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, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters, respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an upper-case 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**.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because 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 the quit, switch, and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The primary *ioctl*(2) system calls have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are:

TCGETA
: Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.

TCSETA
: Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.

TCSETAW
: Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.

TCSETAF
: Wait for the output to drain, then flush the input queue and set the new parameters.

Additional *ioctl*(2) calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK
: Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).

TCXONC
: Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.

TCFLSH
: If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

FILES

/dev/tty*

SEE ALSO

stty(1) in the *User's Reference Manual*.
fork(2), ioctl(2), setpgrp(2), signal(2) in the *Programmer's Reference Manual*.

NAME

termios – general terminal interface

SYNOPSIS

#include <termios.h>

DESCRIPTION

This section describes a general terminal interface for controlling asynchronous communications ports.

When a terminal file is opened, it normally causes the process to wait until the connection is established. In practice, user programs seldom open these files; **getty**(1M) opens them and they become a user's standard input, output and error files.

The file **/dev/tty** is, in each process, the control terminal associated with the process group of that process. Programs or shell sequences use it to ensure that their messages appear on the terminal, no matter how output is redirected. Also, programs that demand an output file name will accept **/dev/tty**, so it is not necessary to determine which terminal is being used.

Opening a terminal device causes the process to block until the connection is established. If the **O_NONBLOCK** flag is set, **open**(2) will return a file descriptor without waiting for the connection to be established.

A terminal may have a distinguished process group associated with it. Certain characters have special functions on input and/or output. The distinguished process group plays a role in the handling of signal-generating characters.

Shells that support job control can allocate the terminal to different jobs, or process groups, by placing related processes in a single process group and associating this process group with the terminal. A terminal's associated process group may be set or examined by a process in the process group using **tcsetpgrp**(3P) and **tcgetpgrp**(3P).

A terminal may belong to a process as its controlling terminal. If a process which is a session process group leader and does not have a controlling terminal, opens a terminal file not already associated with a process group and the **O_NOCTTY** flag is not set, the terminal associated with the terminal file becomes the controlling terminal for the process and the terminal's distinguished process group is set to the process group of the process.

The controlling terminal is inherited by a child process during a **fork**(2). A process relinquishes its controlling terminal when it changes its process group using **setpgrp**(2) or **setsid**(3). When a controlling process terminates, the distinguished process group of its controlling terminal is set to zero. This allows the terminal to be acquired as a controlling terminal by a new session process group leader.

A terminal device associated with a terminal device file may operate in full-duplex mode, so that characters may arrive even while output is occurring. Each terminal device file has associated with it an input queue, into which incoming characters are placed by the system before being read by a process. The system imposes a limit, {MAX_INPUT}, on the number of bytes that may be stored in the input queue. If {MAX_INPUT} is exceeded, the queue is flushed.

A terminal device file may be in canonical mode or non-canonical mode. The mode of the terminal device file determines the method of input processing.

In canonical mode input processing, terminal input is processed in units of lines. A line is delimited by a newline ('\n') character, and end-of-file (EOF) character or an end-of-line (EOL) character. This means that a read request will not be satisfied until an entire line has been typed or a signal has been received. Also, no matter how many characters are requested by the read, at most one line is returned. It is not necessary to read a whole line at once; any number of characters, even one, may be requested in a read without losing information. {MAX_CANON} is the limit on the number of bytes in a line. If this limit is exceeded, the input buffer is flushed. Erase and kill processing will occur during canonical mode input processing.

In non-canonical mode input processing, input characters are not assembled into line, and erase and kill processing does not occur. The values of the special characters **MIN** and **TIME** are used to determine how to process the characters received. **MIN** and **TIME** are defined in the **c_cc** array of special control characters.

MIN represents the minimum number of characters that should be received when the read is satisfied. **TIME** is a timer of 0.1 second granularity that is used to time-out data characterized by short bursts and short term data transmissions. The four possible combinations for **MIN** and **TIME** are as follows:

MIN > 0, TIME > 0:

In this case, **TIME** serves as an intercharacter timer and is activated after the first character is received. Since it is an intercharacter timer, it is reset after a character is received. When the first character is received, the intercharacter timer is started. If **MIN** characters are received before the timer expires, 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 is returned because the timer is not started unless a character has been received. In this case, the read blocks until the **MIN** and **TIME** mechanisms are activated by the receipt of a character.

MIN > 0, TIME = 0:
When 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. A program that sets **TIME** to zero when reading record-based terminal I/O may block indefinitely on a read operation.

MIN = 0, TIME > 0:
When **MIN** is zero, **TIME** no longer represents an intercharacter timer. **TIME** now serves as a read timer that is activated as soon as the **read**() is processed. A read is satisfied as soon as a single character is received or the read timer expires. Note that if the timer expires, no character will be returned. If the timer does not expire, the only way the read can be satisfied is if a character is received. In this case, reads will not block indefinitely waiting for a character; if no character is received within **TIME***0.1 seconds after the read is initiated the read will return zero characters.

MIN = 0, TIME = 0:
The minimum of either the number of characters requested or the number of characters currently available will be returned without waiting for more characters to be input.

Reads are also dependent on the whether the **O_NONBLOCK** flag is set by the **open**(2) or **fcntl**(2) call. If the **O_NONBLOCK** flag is not set, then a read request will block until data is available or a signal is received. If the **O_NONBLOCK** flag is set, then reads will complete without blocking 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 the requested 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 return -1 and **errno** will be set to **EAGAIN**.

Routines that control terminal characteristics do so by modifying the **termios** structure for the device. This structure is defined in **<termios.h>** as follows:

```
struct termios {
    tcflag_t   c_iflag;
    tcflag_t   c_oflag;
    tcflag_t   c_cflag;
    tcflag_t   c_lflag;
    char           c_line;
    unsigned char  c_cc[NCCS];
};
```

The *c_iflag* field describes the basic terminal input control.

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IXON	0002000	Enable start/stop output control.
IXOFF	0010000	Enable start/stop input control.
IUCLC	0001000	Map uppercase to lowercase on input.
IXANY	0004000	Enable any character to restart output.

If **IGNBRK** is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if **BRKINT** is set, the break condition will generate an interrupt signal and flush both the input and output queues. If neither **IGNBRK** or **BRKINT** is set, the break condition is read as NUL (0). If **IGNPAR** is set, characters with other framing and parity errors are ignored.

If **PARMRK** is set, a character with a framing or parity error which is not ignored is read as the three-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if **ISTRIP** is not set, a valid character of 0377 is read as 0377, 0377. If **PARMRK** is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If **INPCK** is set, input parity checking is enabled. If **INPCK** is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

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 RETURN character. If **IGNCR** is set, a received RETURN character is ignored (not read). If **ICRNL** is set, a received RETURN character is translated in an **NL** character.

If **IUCLC** is 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 will suspend output and a received **START** character will restart output. All start/stop characters are ignored and not read. If **IXANY** is set, any input character will restart output which has been suspended.

If **IXOFF** is set, the system will transmit **START/STOP** characters when the input queue is nearly empty/full.

The initial input control value is all-bits-clear.

The **c_oflag** field specifies the system treatment of output.

OPOST	**0000001**	Postprocess output.
OLCUC	**0000002**	Map lower case to upper on output.
ONLCR	**0000004**	Map NL to CR-NL on output.
OCRNL	**0000010**	Map CR to NL on output.
ONOCR	**0000020**	No CR output at column 0.
ONLRET	**0000040**	NL performs CR function.
OFILL	**0000100**	Use fill characters for delay.
OFDEL	**0000200**	Fill is DEL, else NNUL.
NLDLY	**0000400**	Select newline delays:
NL0	**0**	
NL1	**0000400**	
CRDLY	**003000**	**Select carriage-return delays:**
CR0	**0**	
CR1	**0001000**	
CR2	**0002000**	
CR3	**0003000**	
TABDLY	**0014000**	**Select horizontal-tab delays:**

TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form feed delays:
FF0	0	
FF1	0100000	

If **OPOST** is set, output characters are postprocessed 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. Otherwise the **NL** character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also 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. In all cases a value of 0 indicates no delay. If **OFILL** is set, fill characters will be 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, the carriage return delays are used instead of the newline delays. If **OFILL** is set, two fill characters will be transmitted.

Carriage return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If **OFILL** is set, delay type 1 transmits two fill characters, and type 2, four fill characters.

Horizontal tab delay type 1 is dependent 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 will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If **OFILL** is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The **c_cflag** field describes the hardware control of the terminal.

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
B19200	0000016	19200 baud
B38400	0000017	38400 baud
EXTA	0000016	External A
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.

CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.
LOBLK	0010000	Block layer output.

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, two stops bits are required.

If **PARENB** is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the **PARODD** flag specifies odd parity if set, otherwise even parity is used.

If **CREAD** is set, the receiver is enabled. Otherwise no characters will be received.

If **HUPCL** is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If **CLOCAL** is set, the line is assumed to be a local, direct connection with no modem control. Otherwise modem control is assumed.

If **LOBLK** is set, the output of a job control layer will be blocked when it is not the current layer. Otherwise the output generated by that layer will be multiplexed onto the current layer.

The initial hardware control value after open is **B300,CS8, CREAD, HUPCL.**

The **c_lflag** field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.
TOSTOP	0000400	Send **SIGTTOU** for background output.
XCAS	0000004	Canonical upper/lower presentation.

If **ISIG** is set, each input character is checked against the special control characters **INTR, SUSP, SWTCH** and **QUIT**. If an input character matches one of these control characters, the function associated with that character is performed. If **ISIG** is not set, no checking is done. Thus these special input functions are possible only if **ISIG** is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g., 0377).

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, and EOL. If **ICANON** is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least **MIN** characters have been received or the timeout value **TIME** has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The **MIN** and **TIME** values are stored in the position for the **EOF** and **EOL** characters, respectively. The time value represents tenths of seconds.

If **XCASE** is set, and if **ICANON** is 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**.

If **ECHO** is set, characters are echoed as received.

When **ICANON** is set, the following echo functions are possible. If **ECHO** and **ECHOE** are set, the erase character is echoed as ASCII **BS SP BS**, which will clear the last character from a **CRT** screen. If **ECHOE** is set and **ECHO** is not set, the erase character is echoed as ASCII **SP BS.** If **ECHOK** is set, the **NL** character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If **ECHONL** is set, the **NL** character will be echoed even if **ECHO** is not set. This is useful for terminals set to local echo (so-called half duplex).

Unless escaped, the **EOF** character is not echoed. Because **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 the quit, suspend, switch and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The special characters, their default values and functions are as follows:

INTR	Control-c. If the **ISIG** flag is enabled, generates a **SIGINT** signal which is sent to all processes in the distinguished process group associated with the terminal.
QUIT	ASCII FS. If the **ISIG** flag is enabled, generates a **SIGQUIT** signal which is sent to all process in the distinguished process group associated with the terminal.
ERASE	DELETE. If the **ICANON** flag is set, erases the preceding character. It will not erase beyond the start of a line, as delimited by an **NL**, **EOF** or **EOL** character.
KILL	Control-u. If the **ICANON** flag is set, deletes the entire line, as delimited by an **NL**, **EOF** or **EOL** character.
EOF	Control-d. If the **ICANON** flag is set when this character is received, all the characters waiting to be read are immediately passed to the program without waiting for a newline and the **EOF** is discarded. If there are no characters waiting, zero characters will be passed to the program, indicating an end-of-file condition.
NL	ASCII LF. If the **ICANON** flag is set, this character is the line delimiter ('\n'). It cannot be changed.
EOL	ASCII NUL. If the **ICANON** flag is set, this character is an additional line delimiter similar to **NL**.
SUSP	Control-z. If the **ISIG** flag is set, generates a **SIGTSTP** signal which is sent to all processes in the distinguished process group associated with the terminal.
STOP	Control-s. If **IXON** or **IXOFF** flag is set, this character is used to temporarily suspend output. Useful on terminals to prevent output from disappearing before it can be read.

START	Control-q. If **IXON** or **IXOFF** flag is set, this character is used to resume output that has been suspended by a **STOP** character.
SWTCH	Control-z. Used by the shell layering facility, **shl**, to change the current layer to the control layer.

The START and STOP characters cannot be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and SUSP can be changed using **tcsetattr**(3P). ERASE, KILL, and EOF characters may be escaped by preceding the character with a '\'; in this case, no special function is performed.

c_line specifies the line discipline number for the terminal. The basic line discipline number is zero; this is currently the only line discipline supported.

Special control characters are defined by the **c_cc** array in the **termios** structure. The subscript names and descriptions are as follows:

VEOF	EOF character
VEOL	EOL character
VERASE	ERASE character
VINTR	INTR character
VKILL	KILL character
VQUIT	QUIT character
VSUSP	SUSP character
VMIN	MIN character
VTIME	TIME character
VSTART	START character
VSTOP	STOP character

When a modem disconnect is detected by the terminal interface, a **SIGHUP** is sent to all processes in the distinguished process group associated with the terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If **SIGHUP** is ignored or caught, any subsequent read returns an end-of-file indication until the device is closed. Programs that read a terminal file and test for end-of-file can terminate appropriately after a disconnect.

The last process to close a terminal device file shall cause any output to be sent to the device and any input to be discarded. If **HUPCL** is set in the control structure, and the communications port supports a disconnect function, the terminal device will perform a disconnect.

The following functions are provided for controlling the terminal interface:

cfgetispeed(3P)	Return the input baud rate.
cfgetospeed(3P)	Return the output baud rate.
cfsetispeed(3P)	Set the input baud rate.
cfsetospeed(3P)	Set the output baud rate.
tcdrain(3P)	Wait until all written data is transmitted.
tcflow(3P)	Suspend or restart output or input.
tcflush(3P)	Discard data not transmitted.
tcgetattr(3P)	Get terminal attributes.
tcgetpgrp(3P)	Get distinguished process group ID.
tcsendbreak(3P)	Send a break.
tcsetattr(3P)	Set terminal attributes.
tcsetpgrp(3P)	Set distinguished process group ID.

FILES

/dev/tty

SEE ALSO

cfgetospeed(3P), fcntl(2), getty(1M), open(2), tcdrain(3P), tcgetpgrp(3p), tcgetattr(3P), tcsetpgrp(3P).

NAME

timod – Transport Interface cooperating STREAMS module

DESCRIPTION

timod is a STREAMS module for use with the Transport Interface (TI) functions of the Network Services library. The *timod* module converts a set of *ioctl*(2) calls into STREAMS messages that may be consumed by a transport protocol provider that supports the TI. This allows a user to initiate certain TI functions as atomic operations.

The *timod* module must be pushed (see *STREAMS Primer*) onto only a *stream* terminated by a transport protocol provider which supports the TI.

All STREAMS messages, with the exception of the message types generated from the *ioctl* commands described below, will be transparently passed to the neighboring STREAMS module or driver. The messages generated from the following *ioctl* commands are recognized and processed by the *timod* module. The format of the *ioctl* call is:

```
#include <sys/stropts.h>
               -
               -
struct strioctl strioctl;
               -
               -
strioctl.ic_cmd = cmd;
strioctl.ic_timeout = INFTIM;
strioctl.ic_len = size;
strioctl.ic_dp = (char *)buf

ioctl(fildes, I_STR, &strioctl);
```

Where, on issuance, *size* is the size of the appropriate TI message to be sent to the transport provider and on return *size* is the size of the appropriate TI message from the Transport Provider (TP) in response to the issued TI message. *buf* is a pointer to a buffer large enough to hold the contents of the appropriate TI messages. The TI message types are defined in **<sys/tihdr.h>**. The possible values for the *cmd* field are:

TI_BIND
: Bind an address to the underlying transport protocol provider. The message issued to the TI_BIND *ioctl* is equivalent to the TI message type T_BIND_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_BIND_ACK.

TI_UNBIND
: Unbind an address from the underlying transport protocol provider. The message issued to the TI_UNBIND *ioctl* is equivalent to the TI message type T_UNBIND_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_OK_ACK.

TI_GETINFO
: Get the TI protocol specific information from the transport protocol provider. The message issued to the TI_GETINFO *ioctl* is equivalent to the TI message type T_INFO_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_INFO_ACK.

TI_OPTMGMT
: Get, set or negotiate protocol specific options with the transport protocol provider. The message issued to the TI_OPTMGMT *ioctl* is equivalent to the TI message type T_OPTMGMT_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_OPTMGMT_ACK.

FILES

<sys/timod.h>
<sys/tiuser.h>
<sys/tihdr.h>
<sys/errno.h>

SEE ALSO

tirdwr(7)
STREAMS Primer.
STREAMS Programmer's Guide.
NSE Programmer's Guide, Vol. 1.

DIAGNOSTICS

If the *ioctl* system call returns with a value greater than 0, the lower 8 bits of the return value will be one of the TI error codes as defined in **<sys/tiuser.h>**. If the TI error is of type TSYSERR, the next 8 bits of the return value will contain an error as defined in **<sys/errno.h>** (see *intro*(2)).

NAME

tirdwr – Transport Interface read/write interface STREAMS module

DESCRIPTION

tirdwr is a STREAMS module that provides an alternate interface to a transport provider which supports the Transport Interface (TI) functions of the Network Services library (see Section 3N). This alternate interface allows a user to communicate with the transport protocol provider using the *read*(2) and *write*(2) system calls. The *putmsg*(2) and *getmsg*(2) system calls may also be used. However, *putmsg* and *getmsg* can only transfer data messages between user and *stream.*

The *tirdwr* module must only be pushed (see I_PUSH in *streamio*(7)) onto a *stream* terminated by a transport protocol provider which supports the TI. After the *tirdwr* module has been pushed onto a *stream,* none of the Transport Interface functions can be used. Subsequent calls to TI functions causes an error on the *stream.* Once the error is detected, subsequent system calls on the *stream* returns an error with *errno* set to EPROTO.

The following are the actions taken by the *tirdwr* module when pushed on the *stream,* popped (see I_POP in *streamio*(7)) off the *stream,* or when data passes through it:

push – When the module is pushed onto a *stream,* it checks any existing data destined for the user to ensure that only regular data messages are present. It ignores any messages on the *stream* that relate to process management, e.g., messages that generate signals to the user processes associated with the *stream.* If any other messages are present, the I_PUSH returns an error with *errno* set to EPROTO.

write – The module takes the following actions on data that originated from a *write* system call:

- All messages with the exception of messages that contain control portions (see the *putmsg* and *getmsg* system calls) are transparently passed onto the module's downstream neighbor.
- Any zero length data messages are freed by the module and are not passed onto the module's downstream neighbor.
- Any messages with control portions generate an error, and any further system calls associated with the *stream* fails with *errno* set to EPROTO.

read – The module takes the following actions on data that originated from the transport protocol provider:

- All messages with the exception of those that contain control portions (see the *putmsg* and *getmsg* system calls) are transparently passed onto the module's upstream neighbor.
- The action taken on messages with control portions are:
  + Messages that represent expedited data generates an error. All further system calls associated with the *stream* fails with *errno* set to EPROTO.
  + Any data messages with control portions will have the control portions removed from the message prior to passing the message on to the upstream neighbor.
  + Messages that represent an orderly release indication from the transport provider will generate a zero length data message, indicating the end of file, which will be sent to the reader of the *stream*. The orderly release message itself will be freed by the module.
  + Messages that represent an abortive disconnect indication from the transport provider will cause all further *write* and *putmsg* system calls to fail with *errno* set to ENXIO. All further *read* and *getmsg* system calls will return zero length data (indicating end of file) once all previous data has been read.
  + With the exception of the above rules, all other messages with control portions will generate an error and all further system calls associated with the *stream* will fail with *errno* set to EPROTO.
- Any zero length data messages are freed by the module and they are passed onto the module's upstream neighbor.

pop – When the module is popped off the *stream* or the *stream* is closed, the module takes the following action:

- If an orderly release indication has been previously received, then an orderly release request is sent to the remote side of the transport connection.

SEE ALSO

streamio(7), timod(7)
intro(2), getmsg(2), putmsg(2), read(2), write(2), intro(3) in the *Programmer's Reference Manual.*
STREAMS Primer
STREAMS Programmer's Guide.
NSE Programmer's Guide, Part 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

console(7), mvme335(7)

NAME

udp – Internet User Datagram Protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

int socket(AF_INET, SOCK_DGRAM, 0);

or

#include <tiuser.h>
#include <sys/tpiaddr.h>
#include <netinet/in.h>

int t_open(path, oflag, info)

DESCRIPTION

UDP is a simple, unreliable datagram protocol which is used for the Internet protocol family.

UDP is one of the protocols supported by *tpimux*(4) . As such, UDP has two user level interface points: sockets and the transport library interface (TLI). The socket semantics follow the BSD 4.2 implementation, while the TLI interface is defined in the *SYSTEM V/68 Programmer's Guide.* Socket address formats follow the BSD model of the **sockaddr** structure. TLI addresses use the **tpiaddr** structure. The **tpiaddr** structure represents the same address form as the **sockaddr** structure except the TLI address does not contain the zero pad bytes. Header files, for both sockets and TLI, are supplied for address manipulation. Refer to *inet*(3N), *gethostent*(3N), *getnetent*(3N) for methods of generating addresses suitable for use in the **sockaddr** or **tpiaddr** structures.

UDP sockets are connectionless, and are normally used with the *sendto*(2) and *recvfrom*(2) calls, though the *connect*(2) call may also be used to fix the destination for future packets (in which case the *send*(2) or *write*(2) system calls may be used).

When using TLI, the path specified in the *t_open*(3N) call will select the protocol desired and allocate a transport endpoint. For UDP service, use the device **/dev/tpimux/udp**. This will allocate a transport endpoint with connectionless (T_CLTS) support. The *t_bind*(3N) function associates a protocol address with the supplied transport endpoint. After allocation of the transport endpoint, data may be sent using the *t_sndudata*(3N) function, and data received using the *t_rcvudata*(3N) function.

UDP address formats are identical to those used by TCP. In particular UDP provides a port identifier in addition to the normal Internet address format. Note that the UDP port space is separate from the TCP port space (i.e., a UDP port may not be connected to a TCP port). In addition broadcast packets may be sent (assuming the underlying network supports this) by using a reserved broadcast address; this address is network-interface dependent.

DEVICES

To access UDP for use with TLI, the device **/dev/tpimux/udp** should be used. Devices are not used when allocating a socket.

SEE ALSO

socket(2), intro(3), t_open(3N), tpimux(7), inet(4).

NAME

intro – introduction to system maintenance procedures

DESCRIPTION

This section outlines certain procedures that will be of interest to those charged with the task of system maintenance. For example, included are discussions of boot procedures, recovery from crashes, file backups.

SEE ALSO

System Administrator's Guide

NAME

autodump – SysV88 automatic dump on panic

DESCRIPTION

The dump facility allows you to write (dump) the system memory to a mass storage device, tape or disk, after a system crash (panic). Dumps can be done automatically or manually, depending on the value of the variable **AUTODUMP**, which is set via *sysgen*(1M) under the section *Kernel and Paging Parameters*. The default is to have the auto-dump disabled with a value of **0**. When the auto-dump feature is enabled with a value of **1**, the system automatically dumps the memory after a system crash.

When the auto-dump feature is not enabled, after a system crash, the system displays the dump routine address and the dump device major and minor number.

NOTE

You must set the **DMPDEV** as specified below or no dump address/device will be displayed or used.

Perform the following steps to initiate a manual dump:

Press the **ABORT** button on the processor card.

Set the pc to the address of the dump routine (displayed after the panic), by using the **rm pc** command.

Set the status register to 3700 by using the **rm sr** command.

Execute the **g** (go) command.

The *sysgen*(1M) parameter, **DMPDEV**, is used to override the default value under the section *Kernel and Paging Parameters*. You must set the **DMPDEV** to some disk or tape device because the default value will not use any device to do a dump. It is set to a hexadecimal value corresponding to the major and minor number for the desired dump **block** device to be used. The dump device can, for instance, be changed to a disk slice by setting the appropriate value in the **DMPDEV** variable, i.e., to set the dump device to **/dev/dsk/m323_0s3**, the value of **DMPDEV** is 0x0703.

NOTE

Do not use the major and minor number of the **character** device.

If auto-dump is enabled, you should use a disk slice as the dump device. Otherwise, if you configure a tape as the dump device, you must keep a tape in the drive at all times in case of a system panic. A message displays if there is *not* enough room in the slice to dump all the information. You set up the dump disk slice up using *sledit*(1M).

NAME

dgmon – run diagnostic phases in firmware

SYNOPSIS

dgn
dgn [unit[=number]]
dgn [unit[=number]] [ph=a[-b]] [rep=n] [ucl] [soak]
l(ist) unit
h(elp)
s(how)

(phase range specified ph=a-b means phases a through b)

DESCRIPTION

The Diagnostic Monitor Utility, *dgmon,* allows diagnostics to be run on the 3B2 Computer in the firmware mode via the system console. The particular diagnostic phases are specified via the **dgn** command. The interface and the entry into the diagnostic mode is discussed in detail in the *System Administrator's Guide.* Diagnostics can be invoked for the entire computer, types of devices (e.g., all ports boards), a specific device (e.g., the system board), or a particular phase or range of phases for the device or device type. Each diagnostic phase and phase description can be found by listing the diagnostic phases for each computer device (such as l sbd). The Monitor will list the 3B2 Computer physical devices with the s(how) command.

Types of diagnostic phases:

Normal-Diagnostics that run every time the computer is powered up

Demand-Diagnostics that run during soak or must be specifically requested

Interactive-Diagnostics that must be specifically requested and may cause loss of stored data or require operator intervention.

(Diagnostic Monitor will deny request for diagnostics on unequipped devices and non-existent phases)

Option definitions:

ucl
unconditional execution, run all specified phases and display all failing results

rep=n
repeat phases(s) n times

ph=a
select individual phase

ph=a-b
select phase range a through b

soak
silently and continuously run normal and demand diagnostics for specified range (default: all of phase table) and for specified repetitions (default: continuous-stopped with keyboard entry)

EXAMPLES

dgn	(full system)
dgn ports	(all ports devices)
dgn sbd 0 ucl	(Unconditional execution)
dgn ports 0	(ports 0 diagnostic)
dgn ports 1 ucl	
dgn sbd ph=3	(phase specification)
dgn sbd ph=1-5	(phase range specification)
dgn sbd soak	(diagnostic system board soak)

Whenever specific phases are requested, the device to be tested must be designated.

DIAGNOSTICS

Improperly typed syntax will generate message or specify the invalid input. The "h" command will generate a listing of the correct syntax for the system board firmware.

NAME

filledt – fill Equipped Device Table (EDT)

SYNOPSIS

filledt

DESCRIPTION

filledt is a firmware-level routine that completes the Equipped Device Table (EDT) for the computer. It compares ID codes gathered for devices and subdevices to those in the look-up tables stored in **/dgn/edt_data** and copies data, such as device names, into the EDT when ID matches between the EDT and the look-up table occur. *filledt* also makes a console terminal check.

filledt has two operating modes: power-up (automatic) and manual.

During the power-up sequence it silently completes the EDT and checks the console. An error message, **EDT FILL FAILED**, appears only if a peripheral device fails to respond to subdevice equipage queries.

In the manual request mode *filledt* completes the EDT, reporting success or failure. It checks for a console terminal and reports the console location and c_cflags values.

FILES

/filledt

SEE ALSO

termio(7)

NAME

:mk – remake the binary system and commands from source code

DESCRIPTION

All source code for the SYSTEM V/88 operating system is distributed in the directory **/usr/src.** The directory tree rooted at **/usr/src** includes source code for the operating system, libraries, commands, miscellaneous data files necessary for the system and procedures to transform this source code into an executable system.

Within the **/usr/src** directory are the **cmd, lib, uts, head,** and **stand** directories, as well as commands to remake the parts of the system found under each of these sub-directories. These commands are named *:mk* and *:mk***dir** where **dir** is the name of the directory to be recreated. Each of these *:mk***dir** commands will rebuild all or part of the directory it is responsible for. The *:mk* command will run each of the other commands in order and thus, recreate the whole system. The *:mk* command is distributed only to source code licensees.

The following describes each command with its associated directory:

:mklib
The **lib** directory contains the source code for the system libraries. The most important of these is the C library. Each library is in its own subdirectory. If any arguments are specified on the *:mklib* command line then only the given libraries will be rebuilt. The argument * causes it to rebuild all libraries found under the **lib** directory.

:mkhead
The **head** directory contains the source code versions of the header files found in the **/usr/include** directory. The *:mkhead* command will install the header files given as arguments. The argument * causes it to install all header files.

:mkuts
The **uts** directory contains the source code for the SYSTEM V/88 operating system. The *:mkuts* command takes no arguments and invokes a series of makefiles that will recreate the operating system.

Associated with the operating system is a set of header files that describe the user interface to the operating system. The source for these header files is found in a sub-directory within the **uts** directory tree. The user-accessible versions of these header files are found in the **/usr/include/sys** directory. The *:mksyshead* command will install these header files into the **/usr/include/sys** directory.

:mkstand
The **stand** directory contains stand-alone commands and boot programs. The *:mkstand* command rebuilds and installs these programs.

:mkcmd
The **cmd** directory contains the source code for all the commands available on the system. There are two types of entries within the **cmd** directory: commands whose source code consists of only one file with one of the following suffixes: **.l**, **.y**, **.c**, **.s**, **.sh**, or a sub-directory that contains the multiple source files that comprise a particular command or subsystem. Each sub-directory is assumed to have a makefile (see *make*(1)) with the name *command*.**mk** that will take care of creating everything associated with that directory and its sub-directories.

The *:mkcmd* command transforms source code into an executable command based upon a set of predefined rules. If the *:mkcmd* command encounters a sub-directory within the **cmd** directory then it will run the makefile found in that sub-directory. If no makefile is found, an error is reported. For single file commands, the predefined rules are dependent on the file's suffix. C programs (**.c**) are compiled by the C compiler and loaded stripped with shared text. Assembly language programs (**.s**) are assembled and loaded stripped. Yacc programs (**.y**) and lex programs (**.l**) are processed by *yacc*(1) and *lex*(1) respectively, before C compilation. Shell programs (**.sh**) are copied to create the command. Each of these operations leaves a command in the **./cmd** directory which is then installed into a user-accessible directory by using **/etc/install**.

The arguments to *:mkcmd* are either command names or subsystem names. The subsystems distributed with SYSTEM V/88 are: **acct**, **sgs** and **sccs**.

For example, the entire sccs subsystem can be rebuilt by:

```
/usr/src/:mkcmd sccs
```

The argument * causes all commands and subsystems to be rebuilt.

Makefiles throughout the system, and particularly in the **cmd** directory, have a standard format. In particular, *:mkcmd* depends on each makefile having target entries for *install* and *clobber*. The *install* target should cause everything over which the makefile has jurisdiction to be built and installed by **/etc/install**. The *clobber* target should cause a complete cleanup of all unnecessary files resulting from the previous invocation.

An effort has been made to separate the creation of a command from source and its installation on the running system. The command **/etc/install** is used by *:mkcmd* and most makefiles to install commands in standard directories on the system. The use of *install* allows maximum flexibility in the administration of the system. The *install* command makes very few assumptions about where a command is located, who owns it, and what modes are in effect. All assumptions may be overridden on invocation of the command, or more permanently by redefining a few variables in *install*. The purpose of *install* is to install a new version of a command in the same place, with the same attributes as the prior version.

In addition, the use of a separate command to perform installation allows for the creation of test systems in other than standard places, easy movement of commands to balance load, and independent maintenance of makefiles.

SEE ALSO

install(1M)
lex(1), make(1), yacc(1) in the *Programmer's Reference Manual*.

NAME

sysreset – SysV88 automatic system reset

DESCRIPTION

The **reboot** facility allows the system to be rebooted via software. The software generates a system reset, which causes the ROM BUG to get control. If the BUG supports *boot on reset* and the auto-boot feature is enabled in the BUG, the system is rebooted automatically. With the more recent BUGs for the MVME134 and the newer MC68030 processor boards, the boot on reset must be enabled with the **ab** BUG command. See the section in the *Software Release Guide* (SRG) *MC68030 Processors: Bug Configuration*. This describes how to do the **ab** command.

For the question, `Boot at power-up only  [Y,N]?`, answer: `N`. This requests the BUG to boot after any reset. The reboot facility is supported on the MVME134, MVME136, MVME141, MVME143, and MVME147 CPU boards. It is *not* supported on the MVME130/131/132 and the MVME140 CPU boards.

You can enable the auto-reboot facility through *sysgen*(1M) by setting the variable **SYSRESET** to a non-zero value. When the auto-reboot feature is enabled, the system automatically reboots (resets) itself after a system crash (panic) and optional auto-dump.

The commands **sysadm reboot**, **init 6**, **uadmin 2 1**, and **uadmin 2 2** are used to initiate a system reboot.

NAME

version – version commands

SYNOPSIS

version

q

DESCRIPTION

The *version* firmware command lists information about the ROM code in the 3B2 Computer system board. It prints data about the ROM issue, date, software load and serial number.

Q(uit) provides a return to the Maintenance Control Program (MCP) null mode (or the Debug Monitor (DEMON), if equipped).

PERMUTED INDEX

mvme355: driver for the MVME355	9-Track Tape Controller	mvme355(7)
advertise a directory for remote	access adv:	adv(1M)
fusage: disk	access profiler	fusage(1M)
copy file systems for optimal	access time dcopy:	dcopy(1M)
acctcon1, acctcon2: connect-time	accounting	acctcon(1M)
acctprc1, acctprc2: process	accounting	acctprc(1M)
etc.: shell procedures for	accounting chargefee, ckpacct,	acctsh(1M)
acctdisk, etc.: overview of	accounting and miscellaneous/	acct(1M)
of accounting and miscellaneous	accounting commands /overview	acct(1M)
diskusg: generate disk	accounting data by user ID	diskusg(1M)
acctmerg: merge or add total	accounting files	acctmerg(1M)
command summary from per-process	accounting records acctcms:	acctcms(1M)
wtmpfix: manipulate connect	accounting records fwtmp,	fwtmp(1M)
killall: kill all	active processes	killall(1M)
sar: sa1, sa2, sadc: system	activity report package	sar(1M)
acctmerg: merge or	add total accounting files	acctmerg(1M)
arp:	Address Resolution Protocol	arp(7)
LP Print Service lpfilter:	administer filters used with the	lpfilter(1M)
print service lpforms:	administer forms used with the LP	lpforms(1M)
Administration SA: devices	administered by System	SA(7)
network listener service	administration nlsadmin:	nlsadmin(1M)
rfadmin: Remote File Sharing	administration	rfadmin(1M)
devices administered by System	Administration SA:	SA(7)
menu interface to do system	administration sysadm:	sysadm(1M)
uadmin:	administrative control	uadmin(1M)
swap: swap	administrative interface	swap(1M)
access adv:	advertise a directory for remote	adv(1M)
fumount: forced unmount of an	advertised resource	fumount(1M)
accept, reject:	allow or prevent LP requests	accept(1M)
fsba: file system block	analyzer	fsba(1M)
br: back up and restore files	and/or file systems	br(1M)
/to maintenance commands and	application programs	intro(1M)
resources rmnttry:	attempt to mount queued remote	rmnttry(1M)
autodump: SysV88	automatic dump on panic	autodump(8)
sysreset: SysV88	automatic system reset	sysreset(8)
finc: fast incremental	backup	finc(1M)
ckbupscd: check file system	backup schedule	ckbupscd(1M)
frec: recover files from a	backup tape	frec(1M)
another xformtrk: convert	bad track list from one format to	xformtrk(1M)
read disk resident manufacturer's	bad track list m323rd:	m323rd(1M)
su:	become superuser or another user	su(1M)
mkcomply: force compliance with a	bill of materials	mkcomply(1M)
cpset: install object files in	binary directories	cpset(1M)
source code :mk: remake the	binary system and commands from	:mk(8)
fsba: file system	block analyzer	fsba(1M)
numbers into file system logical	block numbers /convert track	fblkgen(1M)
sync: update the super	block	sync(1M)

uucleanup: uucp spool	directory cleanup	uucleanup(1M)
chroot: change root	directory for a command	chroot(1M)
adv: advertise a	directory for remote access	adv(1M)
mvdir: move a	directory	mvdir(1M)
type, modes, speed, and line	discipline getty: set terminal	getty(1M)
type, modes, speed, and line	discipline uugetty: set terminal	uugetty(1M)
fusage:	disk access profiler	fusage(1M)
diskusg: generate	disk accounting data by user ID	diskusg(1M)
df: report number of free	disk blocks and i-nodes	df(1M)
driver for the MVME323 ESDI	Disk Controller mvme323: general	mvme323(7)
manager ddefs:	disk definition information	ddefs(1M)
dinit:	disk initializer	dinit(1M)
track list m323rd: read	disk resident manufacturer's bad	m323rd(1M)
du: summarize	disk usage	du(1M)
check a file system on a	diskette checkfsys:	checkfsys(1M)
umountfsys: mount, unmount a	diskette file system mountfsys,	mountfsys(1M)
create a file system on a	diskette makefsys:	makefsys(1M)
/portdisplay: configure or	display *tty* ports	portconfig(1M)
information rmntstat:	display mounted resource	rmntstat(1M)
whodo: who is	doing what	whodo(1M)
dname: Print Remote File Sharing	domain and network names	dname(1M)
enpdl: ENP10-B (MVME330B)	Download Program	enpdl(1M)
any minor device on a STREAMS	driver clone: open	clone(7)
ifenp: Network Interface	Driver for MVME330 Boards	ifenp(7)
if37x: Network Interface	Driver for MVME37x Boards	if37x(7)
ramdisk: general ramdisk	driver for SYSTEM V/88	ramdisk(7)
Controller mvme323: general	driver for the MVME323 ESDI Disk	mvme323(7)
Tape Controller mvme355:	driver for the MVME355 9-Track	mvme355(7)
mvme335: MVME035/335 Serial Port	Driver	mvme335(7)
pty: pseudo-terminal	driver	pty(7)
sxt: pseudo-device	driver	sxt(7)
dcon: control	dual console operation	dcon(1M)
extract error records from	dump errdead:	errdead(1M)
save the core image of a system	dump in a file savecore:	savecore(1M)
autodump: SysV88 automatic	dump on panic	autodump(8)
sledit, msledit: slice table	editor	sledit(1M)
fill Equipped Device Table	(EDT) filledt:	filledt(8)
Program enpdl:	ENP10-B (MVME330B) Download	enpdl(1M)
commands performed for multi-user	environment rc2: run	rc2(1M)
stop the Remote File Sharing	environment rfstop:	rfstop(1M)
filledt: fill	Equipped Device Table (EDT)	filledt(8)
strclean: STREAMS	error logger cleanup program	strclean(1M)
strerr: STREAMS	error logger daemon	strerr(1M)
log: interface to STREAMS	error logging and event tracing	log(7)
errdead: extract	error records from dump	errdead(1M)
errdemon:	error-logging daemon	errdemon(1M)
errpt: process a report of logged	errors	errpt(1M)
general driver for the MVME323	ESDI Disk Controller mvme323:	mvme323(7)
setmnt:	establish mount table	setmnt(1M)
pwconv: install and update	**/etc/shadow**	pwconv(1M)
to STREAMS error logging and	event tracing log: interface	log(7)

file patch:	examine and change bytes in a	patch(1M)
crash:	examine system images	crash(1M)
uuxqt:	execute remote command requests	uuxqt(1M)
errdead:	extract error records from dump	errdead(1M)
ixf: software management package	extraction facility	ixf(1M)
helpadm: make changes to the Help	Facility database	helpadm(1M)
package generation	facility /software management	igf(1M)
management package extraction	facility ixf: software	ixf(1M)
finc:	fast incremental backup	finc(1M)
pwck, grpck: password/group	file checkers	pwck(1M)
dd: convert and copy a	file	dd(1M)
mknod: build special	file	mknod(1M)
file system ff: list	file names and statistics for a	ff(1M)
null: the null	file	null(7)
fuser: identify processes using a	file or file structure	fuser(1M)
examine and change bytes in a	file patch:	patch(1M)
core image of a system dump in a	file savecore: save the	savecore(1M)
from a two- to a one-password	file scheme pwunconv: converts	pwunconv(1M)
rfadmin: Remote	File Sharing administration	rfadmin(1M)
rfudaemon: Remote	File Sharing daemon process	rfudaemon(1M)
names dname: Print Remote	File Sharing domain and network	dname(1M)
rfstop: stop the Remote	File Sharing environment	rfstop(1M)
rfpasswd: change Remote	File Sharing host password	rfpasswd(1M)
nsquery: Remote	File Sharing name server query	nsquery(1M)
script rfuadmin: Remote	File Sharing notification shell	rfuadmin(1M)
unadv: unadvertise a Remote	File Sharing resource	unadv(1M)
/rumountall: mount, unmount Remote	File Sharing resources	rmountall(1M)
rfstart: start Remote	File Sharing	rfstart(1M)
processes using a file or	file structure fuser: identify	fuser(1M)
ckbupscd: check	file system backup schedule	ckbupscd(1M)
fsba:	file system block analyzer	fsba(1M)
fsdb:	file system debugger	fsdb(1M)
file names and statistics for a	file system ff: list	ff(1M)
fstyp: determine	file system identifier	fstyp(1M)
fsinfo: print various	file system information	fsinfo(1M)
/convert track numbers into	file system logical block numbers	fblkgen(1M)
mkfs: construct a	file system	mkfs(1M)
mount, unmount a diskette	file system /umountfsys:	mountfsys(1M)
checkfsys: check a	file system on a diskette	checkfsys(1M)
makefsys: create a	file system on a diskette	makefsys(1M)
fsstat: report	file system status	fsstat(1M)
volcopy: make literal copy of	file system	volcopy(1M)
mount, umount: mount and unmount	file systems and remote resources	mount(1M)
back up and restore files and/or	file systems br:	br(1M)
time dcopy: copy	file systems for optimal access	dcopy(1M)
fsck, dfsck: check and repair	file systems	fsck(1M)
labelit: provide labels for	file systems	labelit(1M)
mount, unmount multiple	file systems /umountall:	mountall(1M)
uucp system uucico:	file transport program for the	uucico(1M)
the scheduler for the uucp	file transport program uusched:	uusched(1M)
uucp directories and permissions	file uucheck: check the	uucheck(1M)

module timod: Transport	Interface cooperating STREAMS	timod(7)
Boards ifenp: Network	Interface Driver for MVME330	ifenp(7)
Boards if37x: Network	Interface Driver for MVME37x	if37x(7)
lo: software loopback network	interface	lo(7)
lp335: MVME335 line printer	interface	lp335(7)
mvme327: MVME327 SCSI Controller	Interface	mvme327(7)
Controller VMEmodule	Interface /Communication	mvme332xt(7)
Tape Controller VMEmodule	Interface /MVME350 Streamer	mvme350(7)
MVMETAPE Generic Streamer Tape	Interface mvmetape:	mvmetape(7)
STREAMS module tirdwr: Transport	Interface read/write interface	tirdwr(7)
Transport Interface read/write	interface STREAMS module tirdwr:	tirdwr(7)
swap: swap administrative	interface	swap(1M)
termio: general terminal	interface	termio(7)
administration sysadm: menu	interface to do system	sysadm(1M)
logging and event tracing log:	interface to STREAMS error	log(7)
tty: controlling terminal	interface	tty(7)
multiplexer module ip:	Internet IP protocol streams	ip(7)
Protocol tcp:	Internet Transmission Control	tcp(7)
udp:	Internet User Datagram Protocol	udp(7)
commands and application/ intro:	introduction to maintenance	intro(1M)
intro:	introduction to special files	intro(7)
maintenance procedures intro:	introduction to system	intro(8)
ncheck: generate path names from	i-numbers	ncheck(1M)
streamio: STREAMS	ioctl commands	streamio(7)
module ip: Internet	IP protocol streams multiplexer	ip(7)
killall:	kill all active processes	killall(1M)
labelit: provide	labels for file systems	labelit(1M)
terminal type, modes, speed, and	line discipline getty: set	getty(1M)
terminal type, modes, speed, and	line discipline uugetty: set	uugetty(1M)
lp335: MVME335	line printer interface	lp335(7)
port_hold: hold serial	lines open	port_hold(1M)
directories link, unlink:	link and unlink files and	link(1M)
for a file system ff:	list file names and statistics	ff(1M)
xformtrk: convert bad track	list from one format to another	xformtrk(1M)
resident manufacturer's bad track	list m323rd: read disk	m323rd(1M)
nlsadmin: network	listener service administration	nlsadmin(1M)
volcopy: make	literal copy of file system	volcopy(1M)
newgrp:	log in to a new group	newgrp(1M)
errpt: process a report of	logged errors	errpt(1M)
strclean: STREAMS error	logger cleanup program	strclean(1M)
strerr: STREAMS error	logger daemon	strerr(1M)
log: interface to STREAMS error	logging and event tracing	log(7)
track numbers into file system	logical block numbers /convert	fblkgen(1M)
lo: software	loopback network interface	lo(7)
lpsched, etc.: start/stop the	LP Print Service and move/	lpsched(1M)
lpadmin: configure the	LP Print Service	lpadmin(1M)
administer filters used with the	LP Print Service lpfilter:	lpfilter(1M)
administer forms used with the	LP print service lpforms:	lpforms(1M)
accept, reject: allow or prevent	LP requests	accept(1M)
intro: introduction to	maintenance commands and/	intro(1M)
intro: introduction to system	maintenance procedures	intro(8)

prtconf:	print system configuration	prtconf(1M)
names id:	print user and group IDs and	id(1M)
information fsinfo:	print various file system	fsinfo(1M)
lp335: MVME335 line	printer interface	lp335(7)
lpusers: set	printing queue priorities	lpusers(1M)
lpusers: set printing queue	priorities	lpusers(1M)
bcheckrc: system initialization	procedures brc,	brc(1M)
chargefee, ckpacct, etc.: shell	procedures for accounting	acctsh(1M)
to system maintenance	procedures intro: introduction	intro(8)
errpt:	process a report of logged errors	errpt(1M)
acctprc1, acctprc2:	process accounting	acctprc(1M)
init, telinit:	process control initialization	init(1M)
Remote File Sharing daemon	process rfudaemon:	rfudaemon(1M)
powerdown: stop all	processes and turn off the power	powerdown(1M)
killall: kill all active	processes	killall(1M)
structure fuser: identify	processes using a file or file	fuser(1M)
fusage: disk access	profiler	fusage(1M)
prf: operating system	profiler	prf(7)
prfsnap, prfpr: UNIX system	profiler prfld, prfstat, prfdc,	profiler(1M)
ENP10-B (MVME330B) Download	Program enpdl:	enpdl(1M)
uucico: file transport	program for the **uucp** system	uucico(1M)
m332xctl: MVME332XT control	program	m332xctl(1M)
m336ctl: MVME336 control	program	m336ctl(1M)
m350ctl: MVME350 control	program	m350ctl(1M)
m355ctl: MVME355 control	program	m355ctl(1M)
STREAMS error logger cleanup	program strclean:	strclean(1M)
for the uucp file transport	program uusched: the scheduler	uusched(1M)
commands and application	programs /to maintenance	intro(1M)
arp: Address Resolution	Protocol	arp(7)
module ip: Internet IP	protocol streams multiplexer	ip(7)
Internet Transmission Control	Protocol tcp:	tcp(7)
udp: Internet User Datagram	Protocol	udp(7)
labelit:	provide labels for file systems	labelit(1M)
sxt:	pseudo-device driver	sxt(7)
pty:	pseudo-terminal driver	pty(7)
Remote File Sharing name server	query nsquery:	nsquery(1M)
lpusers: set printing	queue priorities	lpusers(1M)
rmount:	queue remote resource mounts	rmount(1M)
rumount: cancel	queued remote resource request	rumount(1M)
rmnttry: attempt to mount	queued remote resources	rmnttry(1M)
ramdisk: general	ramdisk driver for SYSTEM V/88	ramdisk(7)
bad track list m323rd:	read disk resident manufacturer's	m323rd(1M)
tirdwr: Transport Interface	read/write interface STREAMS/	tirdwr(7)
from per-process accounting	records acctcms: command summary	acctcms(1M)
errdead: extract error	records from dump	errdead(1M)
manipulate connect accounting	records fwtmp, wtmpfix:	fwtmp(1M)
frec:	recover files from a backup tape	frec(1M)
commands from source code :mk:	remake the binary system and	:mk(8)
adv: advertise a directory for	remote access	adv(1M)
uuxqt: execute	remote command requests	uuxqt(1M)
administration rfadmin:	Remote File Sharing	rfadmin(1M)

process rfudaemon:	Remote File Sharing daemon	rfudaemon(1M
network names dname: Print	Remote File Sharing domain and	dname(1M)
rfstop: stop the	Remote File Sharing environment	rfstop(1M)
rfpasswd: change	Remote File Sharing host password	rfpasswd(1M)
query nsquery:	Remote File Sharing name server	nsquery(1M)
shell script rfuadmin:	Remote File Sharing notification	rfuadmin(1M)
unadv: unadvertise a	Remote File Sharing resource	unadv(1M)
/rumountall: mount, unmount	Remote File Sharing resources	rmountall(1M)
rfstart: start	Remote File Sharing	rfstart(1M)
rmount: queue	remote resource mounts	rmount(1M)
rumount: cancel queued	remote resource request	rumount(1M)
and unmount file systems and	remote resources /umount: mount	mount(1M)
rmnttry: attempt to mount queued	remote resources	rmnttry(1M)
Uutry: try to contact	remote system with debugging on	Uutry(1M)
fsck, dfsck: check and	repair file systems	fsck(1M)
fsstat:	report file system status	fsstat(1M)
and i-nodes df:	report number of free disk blocks	df(1M)
errpt: process a	report of logged errors	errpt(1M)
sa1, sa2, sadc: system activity	report package sar:	sar(1M)
cancel queued remote resource	request rumount:	rumount(1M)
reject: allow or prevent LP	requests accept,	accept(1M)
uuxqt: execute remote command	requests	uuxqt(1M)
sysreset: SysV88 automatic system	reset	sysreset(8)
list m323rd: read disk	resident manufacturer's bad track	m323rd(1M)
arp: Address	Resolution Protocol	arp(7)
forced unmount of an advertised	resource fumount:	fumount(1M)
rmntstat: display mounted	resource information	rmntstat(1M)
rmount: queue remote	resource mounts	rmount(1M)
rumount: cancel queued remote	resource request	rumount(1M)
unadvertise a Remote File Sharing	resource unadv:	unadv(1M)
unmount file systems and remote	resources /umount: mount and	mount(1M)
attempt to mount queued remote	resources rmnttry:	rmnttry(1M)
unmount Remote File Sharing	resources /rumountall: mount,	rmountall(1M)
br: back up and	restore files and/or file systems	br(1M)
idload:	RFS user and group mapping	idload(1M)
chroot: change	root directory for a command	chroot(1M)
multi-user environment rc2:	run commands performed for	rc2(1M)
the operating system rc0:	run commands performed to stop	rc0(1M)
dgmon:	run diagnostic phases in firmware	dgmon(8)
dump in a file savecore:	save the core image of a system	savecore(1M)
check file system backup	schedule ckbupscd:	ckbupscd(1M)
transport program uusched: the	scheduler for the uucp file	uusched(1M)
a two- to a one-password file	scheme pwunconv: converts from	pwunconv(1M
File Sharing notification shell	script rfuadmin: Remote	rfuadmin(1M)
mvme327: MVME327	SCSI Controller Interface	mvme327(7)
mvme336: MVME336	Serial Communication Controller	mvme336(7)
port_hold: hold	serial lines open	port_hold(1M)
mvme335: MVME035/335	Serial Port Driver	mvme335(7)
nsquery: Remote File Sharing name	server query	nsquery(1M)
nlsadmin: network listener	service administration	nlsadmin(1M)
/etc.: start/stop the LP Print	Service and move requests	lpsched(1M)

MVMETAPE Generic Streamer	Tape Interface mvmetape:	mvmetape(7)
terminfo/ captoinfo: convert a	termcap description into a	captoinfo(1M)
termio: general	terminal interface	termio(7)
tty: controlling	terminal interface	tty(7)
line discipline getty: set	terminal type, modes, speed, and	getty(1M)
line discipline uugetty: set	terminal type, modes, speed, and	uugetty(1M)
tic:	terminfo compiler	tic(1M)
a termcap description into a	terminfo description /convert	captoinfo(1M)
infocmp: compare or print out	terminfo descriptions	infocmp(1M)
acctmerg: merge or add	total accounting files	acctmerg(1M)
strace: print STREAMS	trace messages	strace(1M)
STREAMS error logging and event	tracing log: interface to	log(7)
another xformtrk: convert bad	track list from one format to	xformtrk(1M)
disk resident manufacturer's bad	track list m323rd: read	m323rd(1M)
logical block/ fblkgen: convert	track numbers into file system	fblkgen(1M)
tcp: Internet	Transmission Control Protocol	tcp(7)
STREAMS module timod:	Transport Interface cooperating	timod(7)
interface STREAMS module tirdwr:	Transport Interface read/write	tirdwr(7)
uucp system uucico: file	transport program for the	uucico(1M)
the scheduler for the uucp file	transport program uusched:	uusched(1M)
debugging on Uutry:	try to contact remote system with	Uutry(1M)
portdisplay: configure or display	*tty* ports portconfig,	portconfig(1M)
powerdown: stop all processes and	turn off the power	powerdown(1M
scheme pwunconv: converts from a	two- to a one-password file	pwunconv(1M)
discipline getty: set terminal	type, modes, speed, and line	getty(1M)
discipline uugetty: set terminal	type, modes, speed, and line	uugetty(1M)
resource unadv:	unadvertise a Remote File Sharing	unadv(1M)
prfstat, prfdc, prfsnap, prfpr:	UNIX system profiler prfld,	profiler(1M)
link, unlink: link and	unlink files and directories	link(1M)
mountfsys, umountfsys: mount,	unmount a diskette file system	mountfsys(1M)
mount, umount: mount and	unmount file systems and remote/	mount(1M)
mountall, umountall: mount,	unmount multiple file systems	mountall(1M)
fumount: forced	unmount of an advertised resource	fumount(1M)
rmountall, rumountall: mount,	unmount Remote File Sharing/	rmountall(1M)
pwconv: install and	update **/etc/shadow**	pwconv(1M)
sync:	update the super block	sync(1M)
du: summarize disk	usage	du(1M)
id: print	user and group IDs and names	id(1M)
idload: RFS	user and group mapping	idload(1M)
udp: Internet	User Datagram Protocol	udp(7)
generate disk accounting data by	user ID diskusg:	diskusg(1M)
su: become superuser or another	user	su(1M)
fuser: identify processes	using a file or file structure	fuser(1M)
file uucheck: check the	uucp directories and permissions	uucheck(1M)
uusched: the scheduler for the	uucp file transport program	uusched(1M)
uucleanup:	uucp spool directory cleanup	uucleanup(1M)
file transport program for the	**uucp** system uucico:	uucico(1M)
config: configure SYSTEM	V/88	config(1M)
general ramdisk driver for SYSTEM	V/88 ramdisk:	ramdisk(7)
fsinfo: print	various file system information	fsinfo(1M)
version:	version commands	version(8)

MOTOROLA INC.

Microcomputer Division
2900 South Diablo Way
Tempe, Arizona 85282
P.O. Box 2953
Phoenix, Arizona 85062

11041 PRINTED IN USA (3/90) WPC 2,500