

DYNIX/ptx Reference Manual
Section 4-8

1003-49234-01

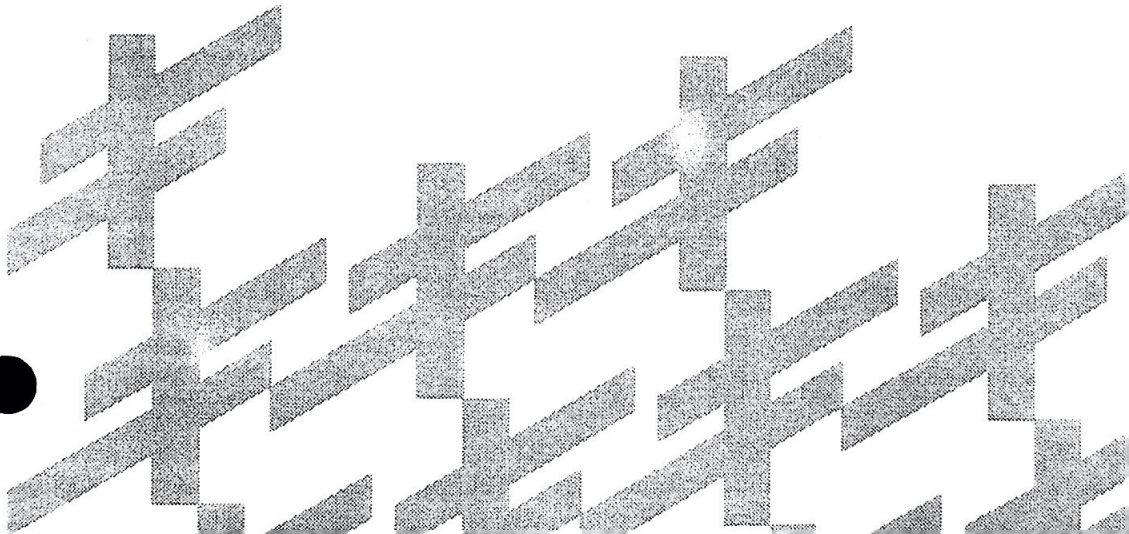


THIS SHEET LIFTER
MUST BE PLACED ON
TOP OF CONTENTS

GRAPHIC PRODUCTS
Chicago, Illinois 60609

DYNIX/ptx®
Reference Manual
Section 4-8

1003-49234-01
Sequent Computer Systems, Inc.



ACT is a trademark of Micro-Term.
Ann Arbor is a trademark of Ann Arbor Terminals.
Beehive is a trademark of Beehive International.
Concept is a trademark of Human Designed Systems.
Dataphone is a registered trademark of AT&T.
DEC is a trademark of Digital Equipment.
Diablo is a registered trademark of Xerox.
Digital is a trademark of Digital Equipment.
DOCUMENTER'S WORKBENCH is a trademark of AT&T.
DYNIX and DYNIX/ptx are registered trademarks of Sequent Computer Systems, Inc.
HP is a trademark of Hewlett-Packard.
i386 is a trademark of Intel Corp.
LSI is a trademark of Lear Siegler.
Micro-term and MIME are trademarks of Micro-Term.
MULTIBUS is a trademark of Intel Corp.
PDP is a trademark of Digital Equipment.
Sequent is a trademark of Sequent Computer Systems, Inc.
Symmetry is a registered trademark of Sequent Computer Systems, Inc.
Tektronix and Tektronix 4010 are registered trademarks of Tektronix.
Teleray is a trademark of Research.
Teletype is a registered trademark of AT&T.
TeleVideo is a registered trademark of TeleVideo Systems.
UNIX is a registered trademark of AT&T.
VAX is a trademark of Digital Equipment.
Versatec is a registered trademark of Versatec.
VT100 is a trademark of Digital Equipment.

Copyright © 1989, 1990 by Sequent Computer Systems, Inc. All rights reserved. This document may not be copied or reproduced in any form without permission from Sequent Computer Systems, Inc. Information in this document is subject to change without notice.

Printed in the United States of America.

About This Manual

Overview

The *DYNIX/ptx Reference Manual* is for programmers working in the DYNIX/ptx operating system environment. This manual describes the programming features of the operating system and assumes that the audience is familiar with the operating system. It does not provide a general overview of the operating system or implementation details of the system.

Organization

The *DYNIX/ptx Reference Manual* is contained in five binders organized as follows:

- Section 1 A-F **Commands**
- Section 1 G-Z **Commands**
- Section 2 **System Calls**
  Section 2SEQ **Sequent System Calls**
  Section 3 **Subroutines**
  - Section 3C **C Library Routines**
  - Section 3M **Mathematical Library Routines**
  - Section 3N **Networking Support Utilities**
  - Section 3PPS **Sequent Parallel Programming Library**
  - Section 3S **Standard I/O Library Routines**
  - Section 3SEQ **Sequent I/O Library Routines**
  - Section 3X **Special Routines**
- Section 4 **File Formats**
  Section 5 **Miscellaneous Facilities**
- Section 7 **Special Files**
  Section 8 **Stand-alone Utilities**

Each binder also contains the following items:

- **Table of Contents** that alphabetically lists the contents of the entire manual set
- **Permuted Index** that lists each programming feature in the entire manual set

The permuted index is a list of keywords, given alphabetically in the second of three columns, together with the context in which each keyword is found. The following shows four separate entries illustrating four ways that the **cat** command can be looked up in the permuted index. Note that you can find the **cat** command by looking up "cat," "concatenate," "files," or "print."

```
                    cat: concatenate and print files.....cat(1)
               cat: concatenate and print files..... cat(1)
concatenate and print files.....cat(1)
     concatenate and print files..... cat(1)
```

Keywords are either topical keywords or the names of manual entries themselves. Entries are identified with their section numbers shown in parentheses. The numbers are important because there is duplication of names among the sections. The right column lists the name of the manual page on which each keyword may be found. The left column contains useful information about the keyword.

The following paragraphs briefly describe the other sections of the manual.

Section 1: Commands describes operating system commands and commands that support C and other programming languages.

Section 2: System Calls describes the access to the services provided by the operating system kernel, including the C language interface.

Section 2SEQ: Sequent System Calls describes the access to the Sequent services provided by the operating system kernel, including the C language interface.

Section 3: Subroutines describes the available subroutines. Their binary versions reside in various system libraries in the directories */lib* and */usr/lib*. The introduction to Section 3 describes these libraries and the files in which they are stored.

Section 4: File Formats documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in *a.out*(4). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language structures corresponding to these formats can be found in the directories */usr/include* and */usr/include/sys*.

Section 5: Miscellaneous Facilities contains a variety of descriptions including descriptions of character sets, macro packages, and so forth.

Section 7: Special Files describes various special files that refer to specific hardware peripherals and operating system device drivers.

Section 8: Stand-alone Utilities describes stand-alone procedures.

References to manual pages with numbers other than those above mean that the utility is contained in the appropriate section of another manual. Each section consists of independent entries which are alphabetized, with the exception of the introductions that begin some sections. Section 3 is divided into subsections in alphabetical order by suffix. Some entries may describe several routines, commands, and so forth. In such cases, the entry appears only once, alphabetized under its *primary* name, the name that appears at the upper corners of each manual page.

Each manual page can be viewed online with the **man**(1) command. For example, enter **man shell** to view the **shell** manual page. Enter **man man** for more information about the **man** command.

Notational Conventions

Manual pages are based on a common format. The main headings are described here—not all of them appear in all cases:

- **NAME** gives the name(s) of the entry and briefly states its purpose. This entry is used to create the Permuted Index and the Table of Contents.
- **SYNOPSIS** summarizes the usage of the program being described. Conventions vary in some sections but are generally as follows:

- **Boldface** strings are literals and are to be typed just as they appear.
- *Italic* strings usually represent substitutable argument prototypes and program names found elsewhere in the manual.
- Square brackets [ ] around an argument prototype indicate that the argument is optional. When an argument prototype is given as *name* or *file*, it refers to a *filename*.
- Ellipses ... are used to show that the previous argument prototype may be repeated.
- A final convention is used by the commands themselves. An argument beginning with a minus –, plus +, or equal sign = is often taken to be some sort of flag argument even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with –, +, or =.

- **CONFIGURATION ENTRY** shows how a driver is called out in a system configuration file.
- **DESCRIPTION** describes the entry, including its purpose and how it is used.
- **FIELDS** describe system file or record entries.
- **EXAMPLES** and **BINARY CONFIGURATION** give examples of usage.
- **FILES** list files that are part of a program, created by a program, or related to a program.
- **SEE ALSO** points to related information in other manual pages.
- **DIAGNOSTICS** discuss the diagnostic (error) messages that may be produced. Messages that are self-explanatory are not listed.
- **WARNINGS** contains warnings against improper user actions.
- **NOTES** give hints about the use of the utility.
- **BUGS** describe known bugs in the software.

Contents

1 Commands

intro(1) ..... introduction to maintenance commands and application programs
300(1): 300, 300s ..... handle special functions of DASI 300 and 300s terminals
4014(1) ..... paginator for the Tektronix 4014 terminal
450(1) ..... handle special functions of the DASI 450 terminal
accept(1M): accept, reject ..... allow or prevent LP requests
acct(1M): acctdisk, acctdusg, accton, acctwtmp ..... overview of accounting and miscellaneous accounting commands
acctcms(1M) ..... command summary from per-process accounting records
acctcom(1) ..... search and print process accounting files
acctcon(1M): acctcon1, acctcon2 ..... connect-time accounting
acctmerg(1M) ..... merge or add total accounting files
acctprc(1M): acctprc1, acctprc2 ..... process accounting
acctsh(1M): chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, shutacct, startup, turnacct ..... shell procedures for accounting
addcmdopt(1M) ..... add a command option to a menu
addformopt(1M) ..... add a form/SELECT window option to a menu
addmenuopt(1M) ..... add a menu option to a menu
admin(1) ..... create and administer SCCS files
apropos(1) ..... locate commands by keyword lookup
ar(1) ..... see *C and Language Tools Binder*
as(1) ..... see *C and Language Tools Binder*
asa(1) ..... interpret ASA carriage control characters
assist(1) ..... assistance using operating system commands
astgen(1) ..... program for generating/modifying ASSIST menus or command forms
at(1): at, batch ..... execute commands at a later time
awk(1) ..... pattern scanning and processing language
banner(1) ..... make posters
basename(1): basename, dirname ..... deliver portions of pathnames
bc(1) ..... arbitrary-precision arithmetic language
bcheckrc(1M) ..... system initialization procedures
bdflush(1M) ..... update daemon
bdiff(1) ..... big diff
bfs(1) ..... big file scanner
bomverify(1M) ..... perform an installation consistency check based on the BOM
bootflags(1M) ..... display/change system boot flags
bp(1M) ..... binary patcher
cal(1) ..... print calendar

calendar(1) ..... reminder service
captoinfo(1M) ..... convert a termcap description into a terminfo description
cat(1) ..... concatenate and print files
cb(1) ..... see *C and Language Tools Binder*
cc(1) ..... see *C and Language Tools Binder*
cd(1) ..... change working directory
cdc(1) ..... change the delta commentary of an SCCS delta
cflow(1) ..... see *C and Language Tools Binder*
checkeq(1) ..... check documents for correct eqn syntax.
checkfsys(1M) ..... check a filesystem
chmod(1) ..... change mode
chown(1): chown, chgrp ..... change owner or group
chroot(1M) ..... change root directory for a command
chrtbl(1M) ..... generate character classification and conversion tables
ci(1) ..... check in RCS revisions
ckbupscd(1M) ..... check filesystem backup schedule
clri(1M) ..... clear inode
cmp(1) ..... compare two files
co(1) ..... check out RCS revisions
col(1) ..... filter reverse line-feeds
comb(1) ..... combine SCCS deltas
comm(1) ..... select or reject lines common to two sorted files
compress(1): compress,
uncompress, zcat ..... compress and expand data
config(1M) ..... build system configuration files
coproc(1F): cocreate, cosend,
cocheck, coreceive, codestroy ..... communicate to a process
cp(1): cp, ln, mv ..... copy, link or move files
cpp(1) ..... see *C and Language Tools Binder*
cpio(1) ..... copy file archives in and out
cprs(1) ..... see *C and Language Tools Binder*
cpset(1) ..... copy file and set mode, user, and group IDs
crash(1M) ..... examine system images
cron(1M) ..... clock daemon
crontab(1) ..... user crontab file
crypt(1) ..... encode/decode
csh(1) ..... a shell (command interpreter) with C-like syntax
csplit(1) ..... context split
ct(1C) ..... spawn getty to a remote terminal
ctags(1) ..... create a tags file
ctrace(1) ..... see *C and Language Tools Binder*
cu(1C) ..... call another UNIX system
cut(1) ..... cut out selected fields of each line of a file
cxref(1) ..... see *C and Language Tools Binder*
date(1) ..... print and set the date
dbx(1) ..... see *C and Language Tools Binder*
dc(1) ..... desk calculator
dcheck(1M) ..... filesystem directory consistency check
dd(1M) ..... convert and copy a file
delta(1) ..... make a delta (change) to an SCCS file

deroff(1) ..... remove nroff, tbl, and eqn constructs
devbuild(1M): devbuild,
devdestroy ..... build and destroy virtual devices
devnm(1M) ..... device name
df(1M) ..... report number of free disk blocks and inodes
diff(1) ..... differential file comparison
diff3(1) ..... 3-way differential file comparison
dircmp(1) ..... directory comparison
dis(1) ..... see *C and Language Tools Binder*
diskusg(1M) ..... generate disk accounting data by user ID
dltsectbl(1M) ..... refresh security table
dmesg(1M) ..... collect system diagnostic messages to form error log
du(1M) ..... summarize disk usage
dump(1) ..... see *C and Language Tools Binder*
dump(1M) ..... incremental filesystem dump
dumpconf(1M) ..... dump kernel auto-configuration device table
dumpfs(1M) ..... dump filesystem information
ec(1M) ..... forms compiler
echo(1) ..... echo arguments
echo(1F) ..... put string on virtual output
ed(1): ed, red ..... text editor
edc(1M) ..... run an externally described command
edit(1) ..... text editor (variant of ex for casual users)
edquota(1M) ..... edit user quotas
efl(1) ..... Extended Fortran Language
egrep(1) ..... search a file for a pattern using full regular expressions
enable(1): enable, disable ..... enable/disable LP printers
env(1) ..... set environment for command execution
ex(1) ..... text editor
expr(1) ..... evaluate arguments as an expression
factor(1) ..... obtain the prime factors of a number
ff(1M) ..... list filenames and statistics for a filesystem
fgrep(1) ..... search a file for a character string
file(1) ..... determine file type
find(1) ..... find files
format(1M) ..... format disks or perform other defect management functions
fsck(1M) ..... filesystem consistency check and interactive repair
fsirand(1M) ..... install random inode generation numbers
fsplit(1) ..... split f77, ratfor, or efl files
fsstat(1M) ..... report filesystem status
fstyp(1M) ..... determine filesystem identifier
fuser(1M) ..... identify processes using a file or file structure
fwtmp(1M): fwtmp, wtmpfix ..... manipulate connect accounting records
gdev(1G): hpd, erase, hardcopy,
tekset, td ..... graphical device routines and filters
ged(1G) ..... graphical editor
gencc(1M) ..... create a front-end to the cc command
get(1) ..... get a version of an SCCS file
getfrm(1F) ..... returns the current frame number

getitems(1F) ..... returns a list of the currently marked menu items
getopt(1) ..... parse command options
getopts(1): getopts, getoptcvt ..... parse command options
getty(1M) ..... set terminal type, modes, speed, and line discipline
glossary(1) ..... definitions of common operating system terms and symbols
graph(1G) ..... draw a graph
graphics(1G) ..... access graphical and numerical commands
greek(1) ..... select terminal filter
grep(1) ..... search a file for a pattern
groups(1) ..... show group memberships
gutil(1G) ..... graphical utilities
help(1) ..... operating system Help Facility
helpadm(1M) ..... change the Help Facility database
hp(1) ..... handle special functions of Hewlett-Packard terminals
hpio(1) ..... Hewlett-Packard 2645A terminal tape file archiver
icheck(1M) ..... filesystem storage consistency check
id(1M) ..... print user and group IDs and names
ident(1) ..... identify files
indicator(1F) ..... display application specific alarms and/or the working indicator
infocmp(1M) ..... compare or print out terminfo descriptions
init(1M): init, telinit ..... process control initialization
install(1M) ..... install commands
installboot(1M) ..... installs a boot program on a disk
ipcrm(1) ..... remove a message queue, semaphore set, or shared memory ID
ipcs(1) ..... report inter-process communication facilities status
isismgr(1M) ..... ISIS file manager, updates text files in ISIS file format
join(1) ..... relational database operator
kill(1) ..... terminate a process
killall(1M) ..... kill all active processes
ksh(1): ksh, rksh ..... the KornShell, a standard/restricted command and programming language
labelit(1M) ..... provide labels for filesystems
ld(1) ..... see *C and Language Tools Binder*
lex(1) ..... generate programs for simple lexical tasks
line(1) ..... read one line
link(1M): link, unlink ..... link and unlink files and directories
lint(1) ..... see *C and Language Tools Binder*
list(1) ..... see *C and Language Tools Binder*
locate(1) ..... use keywords to identify an operating system command
login(1) ..... sign on
logname(1) ..... get login name
lorder(1) ..... see *C and Language Tools Binder*
lp(1): lp, cancel ..... send/cancel requests to the LP print service
lpadmin(1M) ..... configure the LP print service
lpfilter(1M) ..... administer filters used with the LP print service
lpforms(1M) ..... administer forms used with the LP print service

lpsched(1M): lpsched, lpshut,
lpmove ..... start/stop the LP print service and move requests
lpstat(1) ..... print information about the status of the LP print service
lpusers(1M) ..... set printing queue priorities
ls(1) ..... list contents of directory
m4(1) ..... macro processor
machid(1): i386, ns32000, pdp11,
u3b, u3b2, u3b5, vax ..... get processor type truth value
mail(1): mail, rmail ..... send mail to users or read mail
mailbug(1) ..... submit a bug report or enhancement request
mailx(1) ..... interactive message processing system
make(1) ..... maintain, update, and regenerate groups of programs
makealiases(1M): makealiases,
makewhatis ..... generate man page databases
makedev(1M) ..... make system special files
makekey(1) ..... generate encryption key
man(1) ..... display on-line man pages
mcs(1) ..... manipulate the object file comment section
menu(1M) ..... menu processor
menumsg(1M) ..... route user message to menu message queue
merge(1) ..... three-way file merge
mesg(1) ..... permit or deny messages
message(1F) ..... puts its arguments on message line
mkdir(1) ..... make directories
mkfs(1M) ..... construct a filesystem
mknod(1M) ..... build special file
mkpwdbm(1M) ..... make a database for fast password lookup
mkvtoc(1M) ..... populate VTOC on disks
monitor(1M) ..... monitor system activity
more(1) ..... view a file one full screen at a time
mount(1M): mount, umount ..... mount and unmount filesystems
mountall(1M): mountall,
umountall ..... mount, unmount multiple filesystems
mt(1) ..... magnetic tape manipulating program
mvdir(1M) ..... move a directory
nawk(1) ..... pattern scanning and processing language
ncheck(1M) ..... generate pathnames from i-numbers
newform(1) ..... change the format of a text file
newfs(1M) ..... construct a new file system
newgrp(1M) ..... log in to a new group
news(1) ..... print news items
nice(1) ..... run a command at low priority
nl(1) ..... line numbering filter
nm(1) ..... see *C and Language Tools Binder*
noage(1M): noage, nopff, noswap,
onproc ..... run a command with priority aging, PFF, swapping, or processor migration disabled
nohup(1) ..... run a command immune to hangups and quits
od(1) ..... octal dump
online(1M): online, offline ..... bring processors online and offline

pack(1): pack, pcat, unpack ..... compress and expand files
passmgmt(1M) ..... password files management
passwd(1) ..... change login password and password attributes
paste(1) ..... merge same lines of several files or subsequent lines of one file
pathconv(1F) ..... search Interpreter criteria for filename
pdbx(1) ..... see *C and Language Tools Binder*
pg(1) ..... file perusal filter for CRTs
pmap(1M) ..... display or alter parameters of virtual-to-physical mapping devices
pr(1) ..... print files
prof(1) ..... display profile data
prompt(1M) ..... prompt program
prs(1) ..... print an SCCS file
prtvtoc(1M) ..... print the VTOC of a block device
ps(1) ..... report process status
pwck(1M): pwck, grpck ..... password/group file checkers
pwconv(1M) ..... installs and updates /etc/shadow with information from /etc/passwd
pwd(1) ..... working directory name
pwdhd(1M) ..... daemon to oversee password database creation
pwfix(1M) ..... remove incorrectly formatted password entries
pwunconv(1M) ..... converts from a two to a one password file scheme
quota(1) ..... display disk usage and limits
quotacheck(1M) ..... filesystem quota consistency checker
quotaon(1M): quotaon, quotaoff ..... turn filesystem quotas on and off
ratfor(1) ..... rational Fortran dialect
rc0(1M) ..... run commands performed to stop the operating system
rc2(1M) ..... run commands performed for multi-user environment
rcs(1) ..... change RCS file attributes
rcsdiff(1) ..... compare RCS revisions
rcsintro(1) ..... introduction to RCS commands
rcsmerge(1) ..... merge RCS revisions
readfile(1F): readfile, longline ..... reads file and gets longest line
regcmp(1) ..... regular expression compile
regex(1F) ..... match patterns against a string
reinit(1F) ..... run an initialization file
repquota(1M) ..... summarize quotas for a filesystem
reset(1F) ..... reset the current form field to its default values
restore(1M) ..... incremental filesystem restore
rlog(1) ..... print log messages and other information about RCS files
rm(1): rm, rmdir ..... remove files or directories
rmdel(1) ..... remove a delta from an SCCS file
rmvopt(1M) ..... remove a menu option
run(1F) ..... run an executable
runacct(1M) ..... run daily accounting
sact(1) ..... print current SCCS file editing activity
sag(1G) ..... system activity graph
sar(1) ..... system activity reporter
sar(1M): sa1, sa2, sadc ..... system activity report package

savecore(1M) ..... save a core dump of the operating system
sccsdiff(1) ..... compare two versions of an SCCS file
sccstorcs(1) ..... build RCS file from SCCS file
sdiff(1) ..... side-by-side difference program
sed(1) ..... stream editor
set(1F): set, unset ..... set and unset environment variables in core or in files
setcolor(1F) ..... redefine or create a color
setmnt(1M) ..... establish mount table
setpgrp(1) ..... run command in its own process group
setup(1) ..... initialize system for first user
sh(1): sh, rsh ..... shell, the standard/restricted command programming language
shell(1F) ..... run a command using shell
showcfg(1M) ..... show the configuration of the machine
shutdown(1M) ..... shut down system, change system state
size(1) ..... see *C and Language Tools Binder*
sleep(1) ..... suspend execution for an interval
sort(1) ..... sort and/or merge files
spell(1): spell, hashmake, spellin, hashcheck ..... find spelling errors
spline(1G) ..... interpolate smooth curve
split(1) ..... split a file into pieces
starter(1) ..... information about the operating system for beginning users
stat(1G) ..... statistical network useful with graphical commands
strace(1M) ..... print STREAMS trace messages
strclean(1M) ..... STREAMS error logger cleanup program
strerr(1M) ..... STREAMS error logger daemon
strip(1) ..... see *C and Language Tools Binder*
stty(1) ..... set the options for a terminal
su(1M) ..... become superuser or another user
sum(1) ..... print checksum and block count of a file
swap(1M) ..... swap administrative interface
sweepon(1M): sweepon, sweepoff ..... enable/disable patrol seeks on DCC disks
sync(1M) ..... update the superblock
sysadm(1) ..... menu interface to do system administration
tabs(1) ..... set tabs on a terminal
tail(1) ..... deliver the last part of a file
tar(1) ..... tape file archiver
tasktbl(1M) ..... create security table in shared memory
tee(1) ..... pipe fitting
test(1) ..... condition evaluation command
tfipc(1M) ..... interprocess communication reporter
tfmenush(1M) ..... assign menu system as user login shell
tfnotify(1M) ..... send completion message
tic(1M) ..... terminfo compiler
time(1): time, ptime ..... time a command
timex(1) ..... time a command; report process data and system activity
toc(1G): dtoc, ttoc, vtoc ..... graphical table of contents routines
touch(1) ..... update access and modification times of a file

tplot(1G) ..... graphics filters
tput(1) ..... initialize a terminal or query terminfo database
tr(1) ..... translate characters
true(1): true, false ..... provide truth values
tsort(1) ..... topological sort
tty(1) ..... get the name of the terminal
ttydevname(1) ..... get the device associated with a terminal or the terminal associated with a device
tunefs(1M) ..... tune up an existing filesystem
uadmin(1M) ..... administrative control
umask(1) ..... set file-creation mode mask
uname(1) ..... print name of current system
unget(1) ..... undo a previous get of an SCCS file
uniq(1) ..... report repeated lines in a file
units(1) ..... conversion program
uptime(1) ..... show how long system has been up
usage(1) ..... retrieve a command description and usage examples
uucheck(1M) ..... check the uucp directories and permissions file
uucico(1M) ..... file transport program for the uucp system
uucleanup(1M) ..... uucp spool directory clean-up
uucp(1C): uucp, uulog, uuname ..... UNIX-to-UNIX system copy
uugetty(1M) ..... set terminal type, modes, speed, and line discipline
uusched(1M) ..... the scheduler for the uucp file transport program
uustat(1C) ..... uucp status inquiry and job control
uuto(1C): uuto, uupick ..... public UNIX-to-UNIX system file copy
uutry(1M) ..... try to contact remote system with debugging on
uux(1C) ..... UNIX-to-UNIX system command execution
uuxqt(1M) ..... execute remote command requests
val(1) ..... validate SCCS file
vc(1) ..... version control
vi(1) ..... screen-oriented (visual) display editor based on ex
vmtune(1M) ..... tune system-wide virtual-memory parameters
volcopy(1M) ..... make literal copy of filesystem
vsig(1F) ..... synchronize a co-process with the controlling FMLI object
wait(1) ..... await completion of process
wall(1) ..... write to all users
wc(1) ..... word count
what(1) ..... identify SCCS files
whatis(1) ..... describe what a command is
who(1) ..... who is on the system
whodo(1M) ..... who is doing what
write(1) ..... write to another user
xargs(1) ..... construct argument list(s) and execute command
yacc(1) ..... yet another compiler-compiler

2 System Calls

intro(2) ..... introduction to system calls and error numbers
access(2) ..... determine accessibility of a file
acct(2) ..... enable or disable process accounting
alarm(2) ..... set a process alarm clock
brk(2): brk, sbrk ..... change data segment space allocation
chdir(2) ..... change working directory
chmod(2) ..... change mode of file
chown(2) ..... change owner and group of a file
chroot(2) ..... change root directory
close(2) ..... close a file descriptor
creat(2) ..... create a new file or rewrite an existing one
dup(2) ..... duplicate an open file descriptor
dup2(2) ..... duplicate an open file descriptor
exec(2): execl, execv, execle,
execve, execlp, execvp ..... execute a file
exit(2): exit, _exit ..... terminate process
fcntl(2) ..... file control
fork(2) ..... create a new process
getdents(2) ..... read directory entries and put in a file system independent format
getgroups(2) ..... get supplementary group IDs
getmsg(2) ..... get next message off a stream
getpid(2): getpid, getpgrp, getppid .... get process, process group, and parent process IDs
getuid(2): getuid, geteuid, getgid,
getegid ..... get real user, effective user, real group, and effective group IDs
ioctl(2) ..... control device
kill(2) ..... send a signal to a process or a group of processes
link(2) ..... link to a file
lseek(2) ..... move read/write file pointer
mkdir(2) ..... make a directory
mknod(2): mknod ..... make a directory, or a special or ordinary file
mount(2) ..... mount a filesystem
msgctl(2) ..... message control operations
msgget(2) ..... get message queue
msgop(2) ..... message operations
nice(2) ..... change priority of a process
open(2) ..... open for reading or writing
pathconf(2): pathconf, fpathconf ..... get configurable pathname variables
pause(2) ..... suspend process until signal
pipe(2) ..... create an interprocess channel
plock(2) ..... lock process, text, or data in memory
poll(2) ..... STREAMS input/output multiplexing
profil(2) ..... execution time profile
ptrace(2): ptrace, mptrace ..... process trace facility
putmsg(2) ..... send a message on a stream
read(2) ..... read from file
rename(2) ..... change the name of a file
rmdir(2) ..... remove a directory

semctl(2) ..... semaphore control operations
semget(2) ..... get set of semaphores
semop(2) ..... semaphore operations
setpgid(2) ..... manipulate signal sets
setpgrp(2) ..... set process group ID
setsid(2) ..... create session and set process group ID
setuid(2): setuid, setgid ..... set user and group IDs
shmctl(2) ..... shared memory control operations
shmget(2) ..... get shared memory segment identifier
shmop(2): shmop, shmat, shmdt ..... shared memory operations
sigaction(2) ..... examine and change signal actions
sigemptyset(2): sigemptyset,
sigfillset, sigaddset, sigdelset,
sigismember ..... manipulate signal sets
signal(2) ..... specify what to do upon receipt of a signal
sigpending(2) ..... examine pending signals
sigprocmask(2) ..... examine and change blocked signals
sigset(2): sigset, sighold, sigrelse,
sigignore, sigpause ..... signal management
sigsuspend(2) ..... wait for a signal
stat(2): stat, lstat, fstat ..... get file status
statfs(2): statfs, fstatfs ..... get filesystem information
stime(2) ..... set time
stty(2): stty, gtty ..... set and get terminal state (defunct)
sync(2) ..... update superblock
syscall(2) ..... indirect system call
sysconf(2) ..... get configurable system variables
sysi86(2) ..... machine specific functions
tcgetattr(2): tcgetattr, tcsetattr ..... get/set terminal state
tcgetpgrp(2) ..... get foreground process group ID
tcsetpgrp(2) ..... set foreground process group ID
time(2) ..... get time
times(2) ..... get process and child process times
uadmin(2) ..... administrative control
ulimit(2) ..... get and set user limits
umask(2) ..... set and get file creation mask
umount(2) ..... unmount a filesystem
uname(2) ..... get name of current operating system
unlink(2) ..... remove directory entry
ustat(2) ..... get filesystem statistics
utime(2) ..... set file access and modification times
vfork(2) ..... spawn new process in a virtual memory efficient way
wait(2): wait, waitpid ..... wait for a child process to stop or terminate
write(2) ..... write on a file

2SEQ Sequent System Calls

intro(2SEQ) ..... introduction to Sequent system calls
bdflush(2SEQ) ..... update daemon
fvhangup(2SEQ) ..... virtually hangup the specified controlling terminal
get_process_stats(2SEQ) ..... get resource utilization information
getdtablemax(2SEQ) ..... get maximum in-use descriptor table index
getdtablesize(2SEQ) ..... get descriptor table size
getpagesize(2SEQ) ..... get system page size
mmap(2SEQ) ..... map an open file into the processs address space
munmap(2SEQ) ..... unmap part of the processs address space
proc_ctl(2SEQ) ..... control over various process attributes
quotactl(2SEQ) ..... manipulate disk quotas
readlink(2SEQ) ..... read value of a symbolic link
readv(2SEQ) ..... read vector from file
setdtablesize(2SEQ) ..... set descriptor table size
setgroups(2SEQ) ..... set group access list
sigcontext(2SEQ) ..... get signal context
sigstack(2SEQ) ..... set and/or get signal stack context
symlink(2SEQ) ..... make symbolic link to a file
tmp_affinity(2SEQ) ..... bind a process to a processor
tmp_ctl(2SEQ) ..... multi-processor control
truncate(2SEQ): truncate,
ftruncate ..... truncate a file to a specified length
vm_ctl(2SEQ) ..... examine and change virtual memory tuning parameters
writev(2SEQ) ..... write vector on a file

3 Subroutines

intro(3) ..... introduction to functions and libraries

3C C Library Routines

a64l(3C): a64l, l64a ..... convert between long integer and base-64 ASCII string
abort(3C) ..... generate an IOT fault
abs(3C) ..... return integer absolute value
bsearch(3C) ..... binary search a sorted table
clock(3C) ..... report CPU time used
conv(3C): toupper, tolower,
_toupper, _tolower, toascii ..... translate characters
crypt(3C): crypt, setkey, encrypt ..... generate hashing encryption
ctime(3C): ctime, localtime,
gmtime, asctime, cftime, ascftime,
mktime, tzset ..... convert date and time to string
ctype(3C): isdigit, isxdigit,
islower, isupper, isalpha, isalnum,
isspace, iscntrl, ispunct, isprint,
isgraph, isascii, tolower, toupper,
toascci, _tolower, _toupper,
setchrclass ..... character handling

dev_to_ttyname(3C): dev_to_ttyname, ttyname_to_dev ..... convert between major/minor numbers and tty names
dial(3C) ..... establish an out-going terminal line connection
drand48(3C): drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 ..... generate uniformly distributed pseudorandom numbers
ecvt(3C): ecvt, fcvt, gcvt ..... convert floating-point number to string
end(3C): end, etext, edata ..... last locations in program
fpgetround(3C): fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky ..... IEEE floating-point environment control
frexp(3C): frexp, ldexp, modf ..... manipulate parts of floating-point numbers
ftw(3C) ..... walk a file tree
getcwd(3C) ..... get pathname of current working directory
getenv(3C) ..... return value for environment name
getgrent(3C): getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent ..... get group file entry
getlogin(3C) ..... get login name
getopt(3C) ..... get option letter from argument vector
getpass(3C) ..... read a password
getpw(3C) ..... get name from UID
getpwent(3C): getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent ..... get password file entry
getut(3C): getutent, getutid, getutline, pututline, setutent, endutent, utmpname, getutpid ..... access utmp file entry
hsearch(3C): hsearch, hcreate, hdestroy ..... manage hash search tables
isnan(3C): isnand, isnanf ..... test for floating-point NaN (Not-A-Number)
l3tol(3C): l3tol, ltol3 ..... convert between 3-byte integers and long integers
lockf(3C) ..... record locking on files
lsearch(3C): lsearch, lfind ..... linear search and update
malloc(3C): malloc, free, realloc, calloc ..... main memory allocator
memory(3C): memccpy, memchr, memcmp, memcpy, memset ..... memory operations
mkfifo(3C) ..... make a FIFO special file
mktemp(3C) ..... make a unique filename
monitor(3C) ..... prepare execution profile
nlist(3C) ..... get entries from name list
perror(3C): perror, errno, sys_errlist, sys_nerr ..... system error messages
putenv(3C) ..... change or add value to environment
putpwent(3C) ..... write password file entry
qsort(3C) ..... quicker sort
rand(3C): rand, srand ..... simple random-number generator
setjmp(3C): setjmp, longjmp ..... non-local goto
setlocale(3C) ..... set programs location

sigsetjmp(3C): sigsetjmp,
siglongjmp ..... non-local jumps
sleep(3C) ..... suspend execution for interval
ssignal(3C): ssignal, gsignal ..... software signals
stdipc(3C) ..... standard interprocess communication package
strftime(3C) ..... format time information
string(3C): strcat, strdup, strncat,
strcmp, strncmp, strcpy, strncpy,
strlen, strchr, strrchr, strpbrk,
strspn, strcspn, strtok ..... string operations
strstr(3C) ..... find first occurrence of character
strtod(3C): strtod, atof ..... convert string to double-precision number
strtol(3C): strtol, atol, atoi ..... convert string to integer
swab(3C) ..... swap bytes
tsearch(3C): tsearch, tfind,
tdelete, twalk ..... manage binary search trees
ttyname(3C): ttyname, isatty ..... find name of a terminal
ttyslot(3C) ..... find the slot in the utmp file of the current user

3M Mathematical Library Routines

bessel(3M): j0, j1, jn, y0, y1, yn ..... Bessel functions
erf(3M): erf, erfc ..... error function and complementary error function
exp(3M): exp, log, log10, pow, sqrt .... exponential, logarithm, power, square root functions
floor(3M): floor, ceil, fmod, fabs ..... floor, ceiling, remainder, absolute value functions
gamma(3M) ..... log gamma function
hypot(3M) ..... Euclidean distance function
matherr(3M) ..... error-handling function
sinh(3M): sinh, cosh, tanh ..... hyperbolic functions
trig(3M): sin, cos, tan, asin, acos,
atan, atan2 ..... trigonometric functions

3N Networking Support Utilities

t_accept(3N) ..... accept a connect request
t_alloc(3N) ..... allocate a library structure
t_bind(3N) ..... bind an address to a transport endpoint
t_close(3N) ..... close a transport endpoint
t_connect(3N) ..... establish a connection with another transport user
t_error(3N) ..... produce error message
t_free(3N) ..... free a library structure
t_getinfo(3N) ..... get protocol-specific service information
t_getstate(3N) ..... get the current state
t_listen(3N) ..... listen for a connect request
t_look(3N) ..... look at the current event on a transport endpoint
t_open(3N) ..... establish a transport endpoint
t_optmgmt(3N) ..... manage options for a transport endpoint
t_rcv(3N) ..... receive data or expedited data sent over a connection
t_rcvconnect(3N) ..... receive the confirmation from a connect request
t_rcvdis(3N) ..... retrieve information from disconnect
t_rcvrel(3N) ..... acknowledge receipt of an orderly release indication

t_rcvudata(3N) ..... receive a data unit
t_rcvuderr(3N) ..... receive a unit data error indication
t_snd(3N) ..... send data or expedited data over a connection
t_snddis(3N) ..... send user-initiated disconnect request
t_sndrel(3N) ..... initiate an orderly release
t_sndudata(3N) ..... send a data unit
t_sync(3N) ..... synchronize transport library
t_unbind(3N) ..... disable a transport endpoint

3PPS Sequent Parallel Programming Library

intro(3PPS) ..... introduction to Parallel Programming Library
brk(3PPS): brk, sbrk ..... change private data segment size
cpus_online(3PPS) ..... returns the number of CPUs on-line
m_fork(3PPS) ..... execute a subprogram in parallel
m_get_myid(3PPS) ..... return process identification
m_get_numprocs(3PPS) ..... return number of child processes
m_kill_procs(3PPS) ..... kill child processes
m_lock(3PPS): m_lock, m_unlock ..... lock, unlock locks
m_next(3PPS) ..... increment global counter
m_park_procs(3PPS): m_park_procs,
m_rele_procs ..... suspend and resume child process execution
m_set_procs(3PPS) ..... set number of child processes
m_single(3PPS): m_single,
m_multi ..... mark single-process code section
m_sync(3PPS) ..... check in at barrier
s_lock(3PPS): s_init_lock, s_lock,
s_clock, s_unlock ..... initialize, lock, unlock locks
s_wait_barrier(3PPS): s_init_barrier,
s_wait_barrier ..... initialize barrier, wait at barrier
shbrk(3PPS): shbrk, shsbrk ..... change shared data segment size
shmalloc(3PPS): shmalloc,
shrealloc, shfree ..... shared memory allocator

3S Standard I/O Library Routines

ctermid(3S) ..... generate filename for terminal
cuserid(3S) ..... get character login name of the user
fclose(3S): fclose, fflush ..... close or flush a stream
ferror(3S): ferror, feof, clearerr,
fileno ..... stream status inquiries
fopen(3S): fopen, freopen, fdopen ..... open a stream
fread(3S): fread, fwrite ..... binary input/output
fseek(3S): fseek, rewind, ftell ..... reposition a file pointer in a stream
getc(3S): getc, getchar, fgetc, getw .... get character or word from a stream
gets(3S): gets, fgets ..... get a string from a stream
popen(3S): popen, pclose ..... initiate pipe to/from a process
printf(3S): printf, fprintf, sprintf ..... print formatted output

putc(3S): putc, putchar, fputc,
putw ..... put character or word on a stream
puts(3S): puts, fputs ..... put a string on a stream
scanf(3S): scanf, fscanf, sscanf ..... convert formatted input
setbuf(3S): setbuf, setvbuf ..... assign buffering to a stream
stdio(3S) ..... standard buffered input/output package
system(3S) ..... issue a shell command
tmpfile(3S) ..... create a temporary file
tmpnam(3S): tmpnam, tempnam ..... create a name for a temporary file
ungetc(3S) ..... push character back into input stream
vprintf(3S): vprintf, vfprintf,
vsprintf ..... print formatted output of a varargs argument list

3SEQ Sequent I/O Library Routines

dis386(3SEQ) ..... disassemble Series 386 instructions
getgeombyname(3SEQ) ..... get disk geometry by name
getmntent(3SEQ): addmntent,
endmntent, getmntent,
hasmntopt, setmntent ..... get file system descriptor file entry
getpseudotty(3SEQ) ..... get pseudo terminals
getscsiinfo(3SEQ): getscsiinfo,
getscsimatch ..... get scsiinfo structure
getusclk(3SEQ): getusclk,
usclk_init ..... get/init microsecond clock
getzdinfo(3SEQ): getzdinfo,
getzdinfobyname ..... get zdinfo structure
initgroups(3SEQ) ..... initialize group access list
nblocks(3SEQ) ..... calculate number of blocks and indirect blocks
offline_all(3SEQ) ..... take all but one active processor off line
read_constab(3SEQ) ..... read constab entries into a structure
valloc(3SEQ) ..... aligned memory allocator

3X Special Routines

assert(3X) ..... verify program assertion
crypt(3X) ..... password and file encryption functions
curses(3X) ..... terminal screen handling and optimization package
directory(3X): opendir, readdir,
telldir, seekdir, rewinddir, closedir .... directory operations
getspent(3X): getspent, getspnam,
setspent, endspent, fgetspent,
lckpwdf, ulckpwdf ..... get shadow password file entry
ldahread(3X) ..... read the archive header of a member of an archive file
ldclose(3X): ldclose, ldaclose ..... close a common object file
ldfhread(3X) ..... read the file header of a common object file
ldgetname(3X) ..... retrieve symbol name for common object file symbol table entry

ldlread(3X): ldlread, ldlinit, ldlitem .. manipulate line number entries of a common object file function
ldlseek(3X): ldlseek, ldnlseek..... seek to line number entries of a section of a common object file
ldohseek(3X) ..... seek to the optional file header of a common object file
ldopen(3X): ldopen, ldaopen ..... open a common object file for reading
ldrseek(3X): ldrseek, ldnrseek ..... seek to relocation entries of a section of a common object file
ldshread(3X): ldshread, ldnshread .... read an indexednamed section header of a common object file
ldsseek(3X): ldsseek, ldnsseek ..... seek to an indexednamed section of a common object file
ldtbindex(3X) ..... compute the index of a symbol table entry of a common object file
ldtbread(3X) ..... read an indexed symbol table entry of a common object file
ldtbseek(3X) ..... seek to the symbol table of a common object file
logname(3X) ..... return login name of user
malloc(3X): malloc, free, realloc, calloc, mallopt, mallinfo ..... fast main memory allocator
ndbm(3X): dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr, dmb_blkflush ..... database subroutines
plot(3X) ..... graphics interface subroutines
putspent(3X) ..... write shadow password file entry
regcmp(3X): regcmp, regex ..... compile and execute regular expression
sputl(3X): sputl, sgetl ..... access long integer data in a machine-independent fashion
utadduser(3X): ut_add_user, ut_delete_user ..... add/delete users in utmp file

4 File Formats

intro(4) ..... introduction to file formats
a.out(4) ..... common assembler and link editor output
acct(4) ..... per-process accounting file format
addusr_def(4) ..... add user defaults
alias(4) ..... alias file for FACE
ar(4) ..... see *C and Language Tools Binder*
backup_codes(4) ..... backup cycle code definitions
backup_codes1(4) ..... backup subcode definitions
backup_dev(4) ..... backup devices
backup_info(4) ..... backup set definitions
cftime(4) ..... language specific strings
checklist(4) ..... list of filesystems processed by fsck and ncheck
constab(4) ..... constructed devices table
core(4) ..... format of core image file
cpio(4) ..... format of cpio archive
cronlog(4) ..... cron history file
crontabs(4) ..... clock daemon file
devtab(4) ..... virtual device table

dir(4) ..... format of directories
dirent(4) ..... file system independent directory entry
diskinfo(4) ..... disk device characteristic database
dump(4): dump, dumpdates ..... incremental dump format
env(4): .environ, .pref, .variables ..... FACE files
errorlog(4) ..... internal error log file
errormsg(4) ..... error message text file
filehdr(4) ..... file header for common object files
fs(4): fs, inode ..... format of file system volume
fspec(4) ..... format specification in text files
fstab(4) ..... file-system-table
gettydefs(4) ..... speed and terminal settings used by getty
gps(4): gps ..... graphical primitive string, format of graphical files
group(4) ..... group file
helplog(4) ..... monitor help actions
init.d(4) ..... system level transition files directory
inittab(4) ..... script for the init process
issue(4) ..... issue identification file
ldfcn(4) ..... common object file access routines
limits(4) ..... file header for implementation-specific constants
linenum(4) ..... line number entries in a common object file
loginlog(4) ..... log of failed login attempts
lptypes(4) ..... table of printer types supported by the menu system
mnttab(4) ..... mounted filesystem table
mod_config(4) ..... format of a module configuration file
motd(4) ..... message-of-the-day file
ott(4) ..... files that hold object architecture information
passwd(4) ..... password file
plot(4) ..... graphics interface
product(4) ..... installed products file
profile(4) ..... setting up an environment at login time
proto(4) ..... prototype job file for at
rc(4) ..... reboot and shutdown command scripts
rcsfile(4) ..... format of RCS file
reloc(4) ..... relocation information for a common object file
sccsfile(4) ..... format of SCCS file
scnhdr(4) ..... section header for a common object file
scr_dump(4) ..... format of curses screen image file
shadow(4) ..... restricted password file
shells(4) ..... shell table
shutdown(4) ..... system shutdown information file
spellhist(4) ..... unmatched spelling words file
sulog(4) ..... log of substitute user events
syms(4) ..... common object file symbol table format
sys_config(4) ..... format of a system configuration file
syskeymap(4) ..... identify keys to internal menu system functions
task(4) ..... global security table
term(4) ..... format of compiled term file
termcap(4) ..... terminal capability data base
terminfo(4) ..... terminal capability database

timezone(4) ..... set default system time zone
unistd(4) ..... file header for symbolic constants
utmp(4): utmp, wtmp ..... utmp and wtmp entry formats
uucp(4) ..... networking support files

5 Miscellaneous

intro(5) ..... introduction to miscellany
ascii(5) ..... map of ASCII character set
environ(5) ..... user environment
fcntl(5) ..... file control options
math(5) ..... math functions and constants
prof(5) ..... profile within a function
regexp(5) ..... regular expression compile and match routines
stat(5) ..... data returned by stat system call
term(5) ..... conventional names for terminals
types(5) ..... primitive system data types
values(5) ..... machine-dependent values
varargs(5) ..... handle variable argument list

7 Special Files

intro(7) ..... introduction to special files
clone(7) ..... open any minor device on a STREAMS driver
cmpt(7) ..... compatibility partitioning driver
cn(7) ..... pseudoconsole driver
consmem(7): consmem, sm, ss ..... console memory drivers
console(7) ..... console interface
kmmem(7): kMBmem, kMWmem ..... MULITBUS memory
ld0(7) ..... STREAMS tty line discipline
log(7) ..... interface to STREAMS error logging and event tracing
lp(7) ..... Systech parallel line printer interface
lpmod(7) ..... parallel printer canonical processing module
mbad(7) ..... MULITBUS adapter
mem(7): mem, kmem ..... core memory
mtio(7) ..... UNIX magnetic tape interface
null(7) ..... the null file
pmap(7) ..... map physical addresses into processs virtual address space
sa(7) ..... devices administered by System Administration
sced(7) ..... SCSI/Ethernet/Diagnostics board
sd(7) ..... SCSI disk controller
sp(7) ..... STREAMS pipe device
ssm(7) ..... System Services Module
st(7) ..... Systech MTI-800/1600/1650 terminal multiplexor driver
streamio(7) ..... STREAMS ioctl commands
streampty(7) ..... STREAMS pseudo terminal driver
streamtty(7) ..... Generic STREAMS tty driver interface
termio(7) ..... general terminal interface
tg(7) ..... SCSI 1/2-inch magnetic tape interface

timod(7) ..... Transport Interface cooperating STREAMS module
tirdwr(7) ..... Transport Interface read/write interface STREAMS module
tm(7) ..... SCSI tape controller
ts(7) ..... SCSI tape controller
tty(7) ..... controlling terminal (gentty) interface
usclk(7) ..... 32-bit microsecond counter
vtoc(7) ..... VTOC partitioning driver
wd(7) ..... SSM SCSI disk controller
wp(7) ..... SSM parallel line printer interface
xs(7) ..... Xylogics 781 terminal multiplexor driver
xt(7) ..... Xylogics 472 MULITBUS magnetic tape interface
zd(7) ..... dual-channel disk controller

8 Stand-Alone Utilities

intro(8) ..... introduction to system standalone utilities
ccsformat(8) ..... standalone embedded SCSI disk formatter
boot(8) ..... run a standalone program or bring up the operating system
copy(8) ..... perform a standalone data transfer
dump(8) ..... standalone memory dump
prtvtoc(8) ..... print the VTOC of a standalone disk device
zdformat(8) ..... standalone disk formatter for dual-channel disk controller

Permuted Index

st: Systech MTI-800/1600/	1650 terminal multiplexor driver.....st(7)
hpio: Hewlett-Packard	2645A terminal tape file archiver.....hpio(1)
of DASI 300 and 300s terminals	300, 300s: handle special functions.....300(1)
handle special functions of DASI	300 and 300s terminals.....300(1)
DASI 300 and 300s terminals 300,	300s: handle special functions of.....300(1)
special functions of DASI 300 and	300s terminals.....300(1)
usclk:	32-bit microsecond counter.....usclk(7)
dis386: disassemble Series	386 instructions.....dis386(3SEQ)
DASI 450 terminal	450: handle special functions of the.....450(1)
handle special functions of the DASI	450 terminal.....450(1)
xt: Xylogics	472 MULITBUS magnetic tape interface.....xt(7)
between long integer and base-	64 ASCII string.....a64l(3C)
xs: Xylogics	781 terminal multiplexor driver.....xs(7)
integer and base-64 ASCII string	a64l, l64a: convert between long.....a64l(3C)
	abort: generate an IOT fault.....abort(3C)
	abs: return integer absolute value.....abs(3C)
abs: return integer	absolute value.....abs(3C)
fabs: floor, ceiling, remainder,	absolute value functions.....floor(3M)
t_accept:	accept a connect request.....t_accept(3N)
requests	accept, reject: allow or prevent LP.....accept(1M)
utime: set file	access and modification times.....utime(2)
touch: update	access and modification times of a file.....touch(1)
file	access: determine accessibility of a.....access(2)
initgroups: initialize group	access list.....initgroups(3SEQ)
setgroups: set group	access list.....setgroups(2SEQ)
machine-independent/ sputl, sgetl:	access long integer data in a.....sputl(3X)
ldfcn: common object file	access routines.....ldfcn(4)
endutent, utmpname, getutpid:	access utmp file entry.....getut(3C)
access: determine	accessibility of a file.....access(2)
acct: enable or disable process	accounting.....acct(2)
acctcon1, acctcon2: connect-time	accounting.....acctcon(1M)
acctprc1, acctprc2: process	accounting.....acctprc(1M)
turnacct: shell procedures for	accounting.....acctsh(1M)
runacct: run daily	accounting.....runacct(1M)
/accton, acctwtmp: overview of	accounting and miscellaneous/.....acct(1M)
of accounting and miscellaneous	accounting commands.....acct(1M)
diskusg: generate disk	accounting data by user ID.....diskusg(1M)
acct: per-process	accounting file format.....acct(4)
acctcom: search and print process	accounting files.....acctcom(1)
acctmerg: merge or add total	accounting files.....acctmerg(1M)
command summary from per-process	accounting records.....acctcms(1M)
fwtmp, wtmpfix: manipulate connect	accounting records.....fwtmp(1M)
accounting	acct: enable or disable process.....acct(2)
format	acct: per-process accounting file.....acct(4)
per-process accounting records	acctcms: command summary from.....acctcms(1M)
accounting files	acctcom: search and print process.....acctcom(1)
accounting	acctcon1, acctcon2: connect-time.....acctcon(1M)

acctcon1, acctcon2: connect-time accounting ..... acctcon(1M)
acctwtmp: overview of accounting/ acctdisk, acctdusg, accton, ... acct(1M)
of accounting and/ acctdisk, acctdusg, accton, acctwtmp: overview ..... acct(1M)
accounting files acctmerg: merge or add total ..... acctmerg(1M)
acctdisk, acctdusg, accton, acctwtmp: overview of accounting and/ ..... acct(1M)
accounting acctprc1, acctprc2: process ..... acctprc(1M)
acctprc1, acctprc2: process accounting ..... acctprc(1M)
acctdisk, acctdusg, accton, acctwtmp: overview of accounting and/ ..... acct(1M)
release indication t_rcvrel: acknowledge receipt of an orderly ..... t_rcvrel(3N)
trig: sin, cos, tan, asin, acos, atan, atan2: trigonometric/ ..... trig(3M)
helplog: monitor help actions ..... helplog(4)
sigaction: examine and change signal actions ..... sigaction(2)
killall: kill all active processes ..... killall(1M)
offline_all: take all but one active processor off line ..... offline_all(3SEQ)
mbad: MULITBUS adapter ..... mbad(7)
addcmdopt: add a command option to a menu ..... addcmdopt(1M)
menu addformopt: add a form/SELECT window option to a ..... addformopt(1M)
addmenuopt: add a menu option to a menu ..... addmenuopt(1M)
acctmerg: merge or add total accounting files ..... acctmerg(1M)
addusr.def: add user defaults ..... addusr_def(4)
putenv: change or add value to environment ..... putenv(3C)
menu addcmdopt: add a command option to a ..... addcmdopt(1M)
ut_add_user, ut_delete_user: add/delete users in utmp file ..... utadduser(3X)
option to a menu addformopt: add a form/SELECT window ... addformopt(1M)
menu addmenuopt: add a menu option to a ..... addmenuopt(1M)
hasmntopt, setmntent: get file/ addmntent, endmntent, getmntent, ... getmntent(3SEQ)
map an open file into the processs address space ..... mmap(2SEQ)
munmap: unmap part of the processs address space ..... munmap(2SEQ)
addresses into processs virtual address space ..... pmap(7)
t_bind: bind an address to a transport endpoint ..... t_bind(3N)
pmap: map physical addresses into processs virtual address space ..... pmap(7)
addusr.def: add user defaults ..... addusr_def(4)
files admin: create and administer SCCS ..... admin(1)
print service lpfilter: administer filters used with the LP ..... lpfilter(1M)
print service lpforms: administer forms used with the LP ..... lpforms(1M)
admin: create and administer SCCS files ..... admin(1)
SA: devices administered by System Administration ..... sa(7)
SA: devices administered by System Administration ..... sa(7)
sysadm: menu interface to do system administration ..... sysadm(1)
uadmin: administrative control ..... uadmin(1M)
uadmin: administrative control ..... uadmin(2)
swap: swap administrative interface ..... swap(1M)
/usr/adm/loginlog: log of failed login attempts ..... loginlog(4)
/onproc: run a command with priority aging, PFF, swapping, or processor/ ..... noage(1M)
alarm: set a process alarm clock ..... alarm(2)
alarm: set a process alarm clock ..... alarm(2)
/display application specific alarms and/or the working indicator ..... indicator(1F)
pathalias: alias file for FACE ..... alias(4)
valloc: aligned memory allocator ..... valloc(3SEQ)
t_alloc: allocate a library structure ..... t_alloc(3N)
brk, sbrk: change data segment space allocation ..... brk(2)
free, realloc, calloc: main memory allocator ..... malloc(3C)

mallopt, mallinfo: fast main memory allocator.....malloc(3X)
shrealloc, shfree: shared memory allocator.....shmalloc(3PPS)
valloc: aligned memory allocator.....valloc(3SEQ)
editor output a.out: common assembler and link .....a.out(4)
to maintenance commands and application programs.....intro(1)
the working/ indicator: display application specific alarms and/or.....indicator(1F)
lookup apropos: locate commands by keyword.....apropos(1)
ar: archive file format.....see *C and Lang Tools Binder*
and libraries ar: maintain archives.....see *C and Lang Tools Binder*
bc: arbitrary-precision arithmetic language .....bc(1)
ott: files that hold object architecture information.....ott(4)
cpio: format of cpio archive .....cpio(4)
the archive header of a member of an archive file.....ldahread(3X)
ar: common archive file format .....see *C and Lang Tools Binder*
archive file ldahread: read the archive header of a member of an .....ldahread(3X)
2645A terminal tape file archiver.....hpio(1)
tar: tape file archiver.....tar(1)
cpio: copy file archives in and out.....cpio(1)
varargs: handle variable argument list .....varargs(5)
print formatted output of a varargs argument list .....vprintf(3S)
xargs: construct argument list(s) and execute command.....xargs(1)
getopt: get option letter from argument vector.....getopt(3C)
echo: echo arguments.....echo(1)
expr: evaluate arguments as an expression .....expr(1)
message: puts its arguments on message line .....message(1F)
bc: arbitrary-precision arithmetic language .....bc(1)
asa: interpret ASA carriage control characters.....asa(1)
characters asa: interpret ASA carriage control.....asa(1)
/localtime, gmtime, asctime, cftime, ascftime, mktime, tzset: convert/.....ctime(3C)
ascii: map of ASCII character set .....ascii(5)
ascii: map of ASCII character set.....ascii(5)
between long integer and base-64 ASCII string .....a64l(3C)
ctime, localtime, gmtime, asctime, cftime, ascftime, mktime, tzset:/ .....ctime(3C)
trig: sin, cos, tan, asin, acos, atan, atan2: trigonometric/.....trig(3M)
as: common assembler.....see *C and Lang Tools Binder*
a.out: common assembler and link editor output.....a.out(4)
assert: verify program assertion.....assert(3X)
assert: verify program assertion.....assert(3X)
system commands assist: assistance using operating.....assist(1)
/program for generating/modifying ASSIST menus or command forms.....astgen(1)
assist: assistance using operating system commands .....assist(1)
generating/modifying ASSIST menus/ astgen: program for.....astgen(1)
later time at, batch: execute commands at a .....at(1)
trig: sin, cos, tan, asin, acos, atan, atan2: trigonometric functions.....trig(3M)
/sin, cos, tan, asin, acos, atan, atan2: trigonometric functions.....trig(3M)
double-precision number strtod, atof: convert string to .....strtod(3C)
strtol, atol, atoi: convert string to integer.....strtol(3C)
strtol, atol, atoi: convert string to integer.....strtol(3C)
change login password and password attributes .....passwd(1)
control over various process attributes .....proc_ctl(2SEQ)
rcs: change RCS file attributes.....rcs(1)
dumpconf: dump kernel auto-configuration device table.....dumpconf(1M)

language awk: pattern scanning and processing.....awk(1)
backup.codes: backup cycle code definitions.....backup_codes(4)
backup.dev: backup devices.....backup_dev(4)
ckbupscd: check filesystem backup schedule .....ckbupscd(1M)
backup.info: backup set definitions.....backup_info(4)
backup.codes1: backup subcode definitions .....backup_codes1(4)
definitions backup.codes: backup cycle code .....backup_codes(4)
definitions backup.codes1: backup subcode.....backup_codes1(4)
backup.dev: backup devices .....backup_dev(4)
backup.info: backup set definitions.....backup_info(4)
banner: make posters.....banner(1)
m_sync: check in at barrier.....m_sync(3PPS)
initialize barrier, wait at barrier.....s_wait_barrier(3PPS)
/s_wait_barrier: initialize barrier, wait at barrier .....s_wait_barrier(3PPS)
of pathnames basename, dirname: deliver portions.....basename(1)
at, batch: execute commands at a later time.....at(1)
language bc: arbitrary-precision arithmetic .....bc(1)
procedures bcheckrc: system initialization.....bcheckrc(1M)
bdflush: update daemon.....bdflush(1M)
bdflush: update daemon.....bdflush(2SEQ)
bdiff: big diff.....bdiff(1)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions .....bessel(3M)
Bessel functions bessel: j0, j1, jn, y0, y1, yn: .....bessel(3M)
bfs: big file scanner.....bfs(1)
fread, fwrite: binary input/output.....fread(3S)
bp: binary patcher.....bp(1M)
bsearch: binary search a sorted table.....bsearch(3C)
tfind, tdelete, twalk: manage binary search trees.....tsearch(3C)
tmp_affinity: bind a process to a processor.....tmp_affinity(2SEQ)
t_bind: bind an address to a transport endpoint.....t_bind(3N)
sum: print checksum and block count of a file.....sum(1)
prtvtoc: print the VTOC of a block device .....prtvtoc(1M)
sigprocmask: examine and change blocked signals.....sigprocmask(2)
number of blocks and indirect blocks.....nblocks(3SEQ)
nblocks: calculate number of blocks and indirect blocks.....nblocks(3SEQ)
df: report number of free disk blocks and inodes.....df(1M)
consistency check based on the BOM.....bomverify(1M)
consistency check based on the BOM bomverify: perform an installation.....bomverify(1M)
bootflags: display/change system boot flags.....bootflags(1M)
installboot: installs a boot program on a disk.....installboot(1M)
bring up the operating system boot: run a standalone program or.....boot(8)
boot flags bootflags: display/change system .....bootflags(1M)
bp: binary patcher.....bp(1M)
allocation brk, sbrk: change data segment space.....brk(2)
segment size brk, sbrk: change private data.....brk(3PPS)
table bsearch: binary search a sorted .....bsearch(3C)
stdio: standard buffered input/output package .....stdio(3S)
setbuf, setvbuf: assign buffering to a stream.....setbuf(3S)
mailbug: submit a bug report or enhancement request .....mailbug(1)
devbuild, devdestroy: build and destroy virtual devices .....devbuild(1M)
sccstorcs: build RCS file from SCCS file.....sccstorcs(1)
mknod: build special file.....mknod(1M)

config: build system configuration files.....config(1M)
swab: swap bytes.....swab(3C)
cc: C compiler.....see *C and Lang Tools Binder*
cflow: generate C flowgraph.....see *C and Lang Tools Binder*
cpp: the C language preprocessor.....see *C and Lang Tools Binder*
cb: C program beautifier.....see *C and Lang Tools Binder*
lint: a C program checker.....see *C and Lang Tools Binder*
cxref: generate a C program cross-reference.....see *C and Lang Tools Binder*
ctrace: C program debugger.....see *C and Lang Tools Binder*
list: produce C source listing.....see *C and Lang Tools Binder*
cal: print calendar.....cal(1)
indirect blocks nblocks: calculate number of blocks and.....nblocks(3SEQ)
dc: desk calculator.....dc(1)
cal: print calendar.....cal(1)
calendar: reminder service.....calendar(1)
stat: data returned by stat system call.....stat(5)
syscall: indirect system call.....syscall(2)
cu: call another UNIX system.....cu(1C)
malloc, free, realloc, calloc: main memory allocator.....malloc(3C)
malloc, free, realloc, calloc, mallopt, mallinfo: fast main memory/.....malloc(3X)
introduction to Sequent system calls.....intro(2SEQ)
intro: introduction to system calls and error numbers.....intro(2)
lp, cancel: send/cancel requests to the LP print service.....lp(1)
LP print service lp, cancel: send/cancel requests to the.....lp(1)
lpmod: parallel printer canonical processing module.....lpmod(7)
description into a terminfo/ captoinfo: convert a termcap.....captoinfo(1M)
asa: interpret ASA carriage control characters.....asa(1)
cat: concatenate and print files.....cat(1)
cb: C program beautifier.....see *C and Lang Tools Binder*
gencc: create a front-end to the cc command.....gencc(1M)
cc: C compiler.....see *C and Lang Tools Binder*
disk formatter CCSformat: standalone embedded SCSI.....ccsformat(8)
cd: change working directory.....cd(1)
an SCCS delta cdc: change the delta commentary of.....cdc(1)
remainder, absolute value/ floor, ceil, fmod, fabs: floor, ceiling,.....floor(3M)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value/.....floor(3M)
cflow: generate C flowgraph...see *C and Lang Tools Binder*
ctime, localtime, gmtime, asctime, cftime, ascftime, mktime, tzset:/.....ctime(3C)
cftime: language specific strings.....cftime(4)
sigprocmask: examine and change blocked signals.....sigprocmask(2)
brk, sbrk: change data segment space allocation.....brk(2)
attributes passwd: change login password and password.....passwd(1)
chmod: change mode.....chmod(1)
chmod: change mode of file.....chmod(2)
putenv: change or add value to environment.....putenv(3C)
chown: change owner and group of a file.....chown(2)
chown, chgrp: change owner or group.....chown(1)
nice: change priority of a process.....nice(2)
brk, sbrk: change private data segment size.....brk(3PPS)
rcs: change RCS file attributes.....rcs(1)
chroot: change root directory.....chroot(2)
chroot: change root directory for a command.....chroot(1M)

shbrk, shsbrk: change shared data segment size ..... shbrk(3PPS)
sigaction: examine and change signal actions ..... sigaction(2)
bootflags: display/ change system boot flags ..... bootflags(1M)
shutdown: shut down system, change system state ..... shutdown(1M)
cdc: change the delta commentary of an SCCS delta ..... cdc(1)
newform: change the format of a text file ..... newform(1)
helpadm: change the Help Facility database ..... helpadm(1M)
rename: change the name of a file ..... rename(2)
delta: make a delta (change) to an SCCS file ..... delta(1)
parameters vm_ctl: examine and change virtual memory tuning ..... vm_ctl(2SEQ)
cd: change working directory ..... cd(1)
chdir: change working directory ..... chdir(2)
pipe: create an interprocess channel ..... pipe(2)
zd: dual- channel disk controller ..... zd(7)
standalone disk formatter for dual- channel disk controller ..... zdformat(8)
strstr: find first occurrence of character ..... strstr(3C)
ungetc: push character back into input stream ..... ungetc(3S)
conversion tables chrtbl: generate character classification and ..... chrtbl(1M)
_tolower, _toupper, setchrclass: character handling ..... ctype(3C)
cuserid: get character login name of the user ..... cuserid(3S)
getc, getchar, fgetc, getw: get character or word from a stream ..... getc(3S)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
ascii: map of ASCII character set ..... ascii(5)
fgrep: search a file for a character string ..... fgrep(1)
diskinfo: disk device characteristic database ..... diskinfo(4)
asa: interpret ASA carriage control characters ..... asa(1)
_tolower, toascii: translate characters ..... conv(3C)
tr: translate characters ..... tr(1)
lastlogin, monacct, nulladm/ chargefee, ckpacct, dodisk, ... acctsh(1M)
chdir: change working directory ..... chdir(2)
filesystem directory consistency check ..... dcheck(1M)
filesystem storage consistency check ..... icheck(1M)
checkfsys: check a filesystem ..... checkfsys(1M)
fsck: filesystem consistency check and interactive repair ..... fsck(1M)
perform an installation consistency check based on the BOM ..... bomverify(1M)
checkeq: check documents for correct eqn syntax ..... checkeq(1)
ckbupscd: check filesystem backup schedule ..... ckbupscd(1M)
m_sync: check in at barrier ..... m_sync(3PPS)
ci: check in RCS revisions ..... ci(1)
co: check out RCS revisions ..... co(1)
permissions file uucheck: check the uucp directories and ..... uucheck(1M)
eqn syntax. checkeq: check documents for correct ..... checkeq(1)
checkfsys: check a filesystem ..... checkfsys(1M)
processed by fsck and ncheck checklist: list of filesystems ..... checklist(4)
sum: print checksum and block count of a file ..... sum(1)
chown, chgrp: change owner or group ..... chown(1)
/m_rele_procs: suspend and resume child process execution ..... m_park_procs(3PPS)
times: get process and child process times ..... times(2)
wait, waitpid: wait for a child process to stop or terminate ..... wait(2)
m_get_numprocs: return number of child processes ..... m_get_numprocs(3PPS)
m_kill_procs: kill child processes ..... m_kill_procs(3PPS)
m_set_procs: set number of child processes ..... m_set_procs(3PPS)

chmod: change mode ..... chmod(1)
chmod: change mode of file ..... chmod(2)
file chown: change owner and group of a ..... chown(2)
chown, chgrp: change owner or group ..... chown(1)
chroot: change root directory ..... chroot(2)
command chroot: change root directory for a ..... chroot(1M)
classification and conversion/ chrtbl: generate character ..... chrtbl(1M)
ci: check in RCS revisions ..... ci(1)
schedule ckbupscd: check filesystem backup ..... ckbupscd(1M)
nulladm, prctmp,/ chargefee, ckpacct, dodisk, lastlogin, monacct, ... acctsh(1M)
chrtbl: generate character classification and conversion tables ..... chrtbl(1M)
clri: clear inode ..... clri(1M)
ferror, feof, clearerr, fileno: stream status inquiries ..... ferror(3S)
a shell (command interpreter) with C-like syntax ..... csh(1)
alarm: set a process alarm clock ..... alarm(2)
usclk_init: get/init microsecond clock ..... getusclk(3SEQ)
cron: clock daemon ..... cron(1M)
crontabs: clock daemon file ..... crontabs(4)
clock: report CPU time used ..... clock(3C)
STREAMS driver clone: open any minor device on a ..... clone(7)
ldclose, ldaclose: close a common object file ..... ldclose(3X)
close: close a file descriptor ..... close(2)
t_close: close a transport endpoint ..... t_close(3N)
close: close a file descriptor ..... close(2)
fclose, fflush: close or flush a stream ..... fclose(3S)
/telldir, seekdir, rewinddir, closedir: directory operations ..... directory(3X)
clri: clear inode ..... clri(1M)
cmp: compare two files ..... cmp(1)
driver cmpt: compatibility partitioning ..... cmpt(7)
cn: pseudoconsole driver ..... cn(7)
co: check out RCS revisions ..... co(1)
communicate to a/ cocreate, cosend, cocheck, coreceive, codestroy: ..... coproc(1F)
coreceive, codestroy: communicate/ cocreate, cosend, cocheck, ... coproc(1F)
backup.codes: backup cycle code definitions ..... backup_codes(4)
m_multi: mark single-process code section ..... m_single(3PPS)
/cosend, cocheck, coreceive, codestroy: communicate to a process ..... coproc(1F)
col: filter reverse line-feeds ..... col(1)
to form error log dmesg: collect system diagnostic messages ..... dmesg(1M)
setcolor: redefine or create a color ..... setcolor(1F)
comb: combine SCCS deltas ..... comb(1)
comb: combine SCCS deltas ..... comb(1)
to two sorted files comm: select or reject lines common ..... comm(1)
mcs: manipulate the object file comment section ..... mcs(1)
cdc: change the delta commentary of an SCCS delta ..... cdc(1)
a.out: common assembler and link editor output ..... a.out(4)
ldclose, ldaclose: close a common object file ..... ldclose(3X)
ldfhread: read the file header of a common object file ..... ldfhread(3X)
number entries of a section of a common object file ..... ldlseek(3X)
to the optional file header of a common object file ..... ldohseek(3X)
relocation entries of a section of a common object file ..... ldrseek(3X)
indexednamed section header of a common object file ..... ldshread(3X)
to an indexednamed section of a common object file ..... ldsseek(3X)

index of a symbol table entry of a common object file ..... ldtbindex(3X)
an indexed symbol table entry of a common object file ..... ldtbread(3X)
seek to the symbol table of a common object file ..... ldtbseek(3X)
linenum: line number entries in a common object file ..... linenum(4)
reloc: relocation information for a common object file ..... reloc(4)
scnhdr: section header for a common object file ..... scnhdr(4)
ldfcn: common object file access routines ..... ldfcn(4)
ldopen, ldaopen: open a common object file for reading ..... ldopen(3X)
/manipulate line number entries of a common object file function ..... ldlread(3X)
ldgetname: retrieve symbol name for common object file symbol table/ ..... ldgetname(3X)
syms: common object file symbol table format ..... syms(4)
filehdr: file header for common object files ..... filehdr(4)
symbols glossary: definitions of common operating system terms and ..... glossary(1)
comm: select or reject lines common to two sorted files ..... comm(1)
cocheck, coreceive, codestroy: communicate to a process ..... coproc(1F)
ipcs: report inter-process communication facilities status ..... ipcs(1)
stdipc: ftok: standard interprocess communication package ..... stdipc(3C)
tfipc: interprocess communication reporter ..... tfipc(1M)
descriptions infocmp: compare or print out terminfo ..... infocmp(1M)
rcsdiff: compare RCS revisions ..... rcsdiff(1)
cmp: compare two files ..... cmp(1)
sccsdiff: compare two versions of an SCCS file ..... sccsdiff(1)
diff: differential file comparison ..... diff(1)
diff3: 3-way differential file comparison ..... diff3(1)
dircmp: directory comparison ..... dircmp(1)
cmpt: compatibility partitioning driver ..... cmpt(7)
regcmp: regular expression compile ..... regcmp(1)
regcmp, regex: compile and execute regular expression ..... regcmp(3X)
regexp: regular expression compile and match routines ..... regexp(5)
term: format of compiled term file ..... term(4)
cc: C compiler ..... see *C and Lang Tools Binder*
ec: forms compiler ..... ec(1M)
tic: terminfo compiler ..... tic(1M)
yacc: yet another compiler- compiler ..... yacc(1)
yacc: yet another compiler-compiler ..... yacc(1)
erf, erfc: error function and complementary error function ..... erf(3M)
cprs: compress an object file ..... see *C and Lang Tools Binder*
compress, uncompress, zcat: compress and expand data ..... compress(1)
pack, pcat, unpack: compress and expand files ..... pack(1)
and expand data compress, uncompress, zcat: compress ..... compress(1)
entry of a common object/ ldtbindex: compute the index of a symbol table ..... ldtbindex(3X)
cat: concatenate and print files ..... cat(1)
test: condition evaluation command ..... test(1)
files config: build system configuration ..... config(1M)
pathconf, fpathconf: get configurable pathname variables ..... pathconf(2)
sysconf: get configurable system variables ..... sysconf(2)
dumpconf: dump kernel auto- configuration device table ..... dumpconf(1M)
mod_config: format of a module configuration file ..... mod_config(4)
sys_config: format of a system configuration file ..... sys_config(4)
config: build system configuration files ..... config(1M)
showcfg: show the configuration of the machine ..... showcfg(1M)
lpadmin: configure the LP print service ..... lpadmin(1M)

t_rcvconnect: receive the — confirmation from a connect request ..... t_rcvconnect(3N)
fwtmp, wtmpfix: manipulate — connect accounting records ..... fwtmp(1M)
t_accept: accept a — connect request ..... t_accept(3N)
t_listen: listen for a — connect request ..... t_listen(3N)
receive the confirmation from a — connect request ..... t_rcvconnect(3N)
establish an out-going terminal line — connection ..... dial(3C)
data or expedited data sent over a — connection ..... t_rcv(3N)
send data or expedited data over a — connection ..... t_snd(3N)
t_connect: establish a — connection with another transport user ..... t_connect(3N)
acctcon1, acctcon2: — connect-time accounting ..... acctcon(1M)
dcheck: filesystem directory — consistency check ..... dcheck(1M)
icheck: filesystem storage — consistency check ..... icheck(1M)
fsck: filesystem — consistency check and interactive repair ..... fsck(1M)
bomverify: perform an installation — consistency check based on the BOM ..... bomverify(1M)
quotacheck: filesystem quota — consistency checker ..... quotacheck(1M)
drivers — consmem, sm, ss: console memory ..... consmem(7)
— console: console interface ..... console(7)
console: — console interface ..... console(7)
consmem, sm, ss: — console memory drivers ..... consmem(7)
— constab: constructed devices table ..... constab(4)
read_constab: read — constab entries into a structure ..... read_constab(3SEQ)
header for implementation-specific — constants ..... limits(4)
math: math functions and — constants ..... math(5)
unistd: file header for symbolic — constants ..... unistd(4)
constab: — constructed devices table ..... constab(4)
deroff: remove nroff, tbl, and eqn — constructs ..... deroff(1)
Uutry: try to — contact remote system with debugging on ..... uutry(1M)
ls: list — contents of directory ..... ls(1)
sigcontext: get signal — context ..... sigcontext(2SEQ)
set and/or get signal stack — context ..... sigstack(2SEQ)
csplit: — context split ..... csplit(1)
sd: SCSI disk — controller ..... sd(7)
tm: SCSI tape — controller ..... tm(7)
ts: SCSI tape — controller ..... ts(7)
wd: SSM SCSI disk — controller ..... wd(7)
zd: dual-channel disk — controller ..... zd(7)
disk formatter for dual-channel disk — controller ..... zdformat(8)
synchronize a co-process with the — controlling FMLI object ..... vsig(1F)
virtually hangup the specified — controlling terminal ..... fvhangup(2SEQ)
tty: — controlling terminal (gentty) interface ..... tty(7)
_tolower, toascii: translate/ — conv: toupper, tolower, _toupper, ... conv(3C)
units: — conversion program ..... units(1)
character classification and — conversion tables ..... chrtbl(1M)
terminfo description captoinfo: — convert a termcap description into a ..... captoinfo(1M)
dd: — convert and copy a file ..... dd(1M)
long integers l3tol, ltol3: — convert between 3-byte integers and ..... l3tol(3C)
base-64 ASCII string a64l, l64a: — convert between long integer and ..... a64l(3C)
dev_to_ttyname, ttyname_to_dev: — convert major/minor numbers ..... dev_to_ttyname(3C)
/cftime, ascftime, mktime, tzset: — convert date and time to string ..... ctime(3C)
ecvt, fcvt, gcvt: — convert floating-point number to string ..... ecvt(3C)
scanf, fscanf, sscanf: — convert formatted input ..... scanf(3S)
strtod, atof: — convert string to double-precision number ..... strtod(3C)

strtol, atol, atoi: convert string to integer ..... strtol(3C)
password file scheme pwunconv: converts from a two to a one ..... pwunconv(1M)
vsig: synchronize a co-process with the controlling FMLI object ..... vsig(1F)
UNIX-to-UNIX system copy ..... uucp(1C)
system file copy ..... uuto(1C)
dd: convert and copy a file ..... dd(1M)
cpset: copy file and set mode, user, and group IDs ..... cpset(1)
cpio: copy file archives in and out ..... cpio(1)
cp, ln, mv: copy, link or move files ..... cp(1)
volcopy: make literal copy of filesystem ..... volcopy(1M)
transfer copy: perform a standalone data ..... copy(8)
savecore: save a core dump of the operating system ..... savecore(1M)
core: format of core image file ..... core(4)
core: format of core image file ..... core(4)
mem, kmem: core memory ..... mem(7)
and unset environment variables in core or in files ..... set(1F)
cocreate, cosend, cocheck, coreceive, codestroy: communicate to a/ ..... coproc(1F)
checkeq: check documents for correct eqn syntax ..... checkeq(1)
trigonometric functions trig: sin, cos, tan, asin, acos, atan, atan2: ..... trig(3M)
codestroy: communicate to/ cocreate, cosend, cocheck, coreceive, ... coproc(1F)
sinh, cosh, tanh: hyperbolic functions ..... sinh(3M)
wc: word count ..... wc(1)
sum: print checksum and block count of a file ..... sum(1)
m_next: increment global counter ..... m_next(3PPS)
usclk: 32-bit microsecond counter ..... usclk(7)
cp, ln, mv: copy, link or move files ..... cp(1)
cpio: format of cpio archive ..... cpio(4)
cpio: copy file archives in and out ..... cpio(1)
cpio: format of cpio archive ..... cpio(4)
cpp: C preprocessor ..... see *C and Lang Tools Binder*
cprs: compress object file ..... see *C and Lang Tools Binder*
and group IDs cpset: copy file and set mode, user, ... cpset(1)
clock: report CPU time used ..... clock(3C)
cpus_online: returns the number of CPUs on-line ..... cpus_online(3PPS)
CPUs on-line cpus_online: returns the number of ..... cpus_online(3PPS)
crash: examine system images ..... crash(1M)
an existing one creat: create a new file or rewrite ..... creat(2)
setcolor: redefine or create a color ..... setcolor(1F)
gencc: create a front-end to the cc command ..... gencc(1M)
tmpnam, tempnam: create a name for a temporary file ..... tmpnam(3S)
creat: create a new file or rewrite an existing one ..... creat(2)
fork: create a new process ..... fork(2)
ctags: create a tags file ..... ctags(1)
tmpfile: create a temporary file ..... tmpfile(3S)
pipe: create an interprocess channel ..... pipe(2)
admin: create and administer SCCS files ..... admin(1)
tasktbl: create security table in shared memory ..... tasktbl(1M)
setsid: create session and set process group ID ..... setsid(2)
pathconv: search Interpreter criteria for filename ..... pathconv(1F)
cron: clock daemon ..... cron(1M)
cronlog: cron history file ..... cronlog(4)
cronlog: cron history file ..... cronlog(4)

crontab: user — crontab file ..... crontab(1)
crontab: user — crontab: user crontab file ..... crontab(1)
crontab: user — crontabs: clock daemon file ..... crontabs(4)
cxref: generate C program — cross-reference ..... see *C and Lang Tools Binder*
pg: file perusal filter for — CRTs ..... pg(1)
crypt: encode/decode ..... crypt(1)
functions — crypt: password and file encryption ..... crypt(3X)
hashing encryption — crypt, setkey, encrypt: generate ..... crypt(3C)
with C-like syntax — csh: a shell (command interpreter) ..... csh(1)
csplit: context split ..... csplit(1)
ct: spawn getty to a remote terminal ..... ct(1C)
ctags: create a tags file ..... ctags(1)
terminal — ctermid: generate filename for ..... ctermid(3S)
cftime, ascftime, mktime, tzset:/ — ctime, localtime, gmtime, asctime, ... ctime(3C)
ctrace: C prog debugger ..... see *C and Lang Tools Binder*
cu: call another UNIX system ..... cu(1C)
scr_dump: format of — curses screen image file ..... scr_dump(4)
optimization package — curses: terminal screen handling and ..... curses(3X)
the user — cuserid: get character login name of ..... cuserid(3S)
line of a file — cut: cut out selected fields of each ..... cut(1)
cut: — cut out selected fields of each line of a file ..... cut(1)
cross-reference — cxref: generate C program ..... see *C and Lang Tools Binder*
backup.codes: backup — cycle code definitions ..... backup_codes(4)
bdflush: update — daemon ..... bdflush(1M)
bdflush: update — daemon ..... bdflush(2SEQ)
cron: clock — daemon ..... cron(1M)
strerr: STREAMS error logger — daemon ..... strerr(1M)
crontabs: clock — daemon file ..... crontabs(4)
creation pwdhd: — daemon to oversee password database ..... pwdhd(1M)
runacct: run — daily accounting ..... runacct(1M)
300s: handle special functions of — DASI 300 and 300s terminals ..... 300(1)
450: handle special functions of the — DASI 450 terminal ..... 450(1)
zcat: compress and expand — data ..... compress(1)
prof: display profile — data ..... prof(1)
time a command; report process — data and system activity ..... timex(1)
termcap: terminal capability — data base ..... termcap(4)
diskusg: generate disk accounting — data by user ID ..... diskusg(1M)
t_rcvuderr: receive a unit — data error indication ..... t_rcvuderr(3N)
sputl, sgetl: access long integer — data in a machine-independent/ ..... sputl(3X)
plock: lock process, text, or — data in memory ..... plock(2)
t_snd: send — data or expedited data over a connection ..... t_snd(3N)
t_rcv: receive — data or expedited data sent over a connection ..... t_rcv(3N)
t_snd: send data or expedited — data over a connection ..... t_snd(3N)
stat: — data returned by stat system call ..... stat(5)
brk, sbrk: change private — data segment size ..... brk(3PPS)
shbrk, shsbrk: change shared — data segment size ..... shbrk(3PPS)
brk, sbrk: change — data segment space allocation ..... brk(2)
t_rcv: receive data or expedited — data sent over a connection ..... t_rcv(3N)
copy: perform a standalone — data transfer ..... copy(8)
types: primitive system — data types ..... types(5)
t_rcvudata: receive a — data unit ..... t_rcvudata(3N)
t_sndudata: send a — data unit ..... t_sndudata(3N)

diskinfo: disk device characteristic database.....diskinfo(4)
helpadm: change the Help Facility database.....helpadm(1M)
terminfo: terminal capability database.....terminfo(4)
a terminal or query terminfo database.....tput(1)
pwdhd: daemon to oversee password database creation.....pwdhd(1M)
mkpwdbm: make a database for fast password lookup.....mkpwdbm(1M)
join: relational database operator.....join(1)
dbm_clearerr, dmb_blkflush: database subroutines.....ndbm(3X)
makewhatis: generate man page databases.....makealiases(1M)
date: print and set the date.....date(1)
ascftime, mktime, tzset: convert date and time to string.....ctime(3C)
date: print and set the date.....date(1)
/dbm_nextkey, dbm_error, dbm_clearerr, dmb_blkflush: database/.....ndbm(3X)
dbm_delete,/ dbm_open, dbm_close, dbm_fetch, dbm_store,.....ndbm(3X)
dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey,/.....ndbm(3X)
/dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr,/.....ndbm(3X)
dbm_firstkey,/dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete,.....ndbm(3X)
/dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error,/.....ndbm(3X)
dbm_store, dbm_delete, dbm_firstkey, dbm_open, dbm_close, dbm_fetch,.....ndbm(3X)
dbm_store, dbm_delete,/ dbm_store, dbm_delete, dbm_firstkey,/.....ndbm(3X)
dbm_open, dbm_close, dbm_fetch, pdbx: parallel debugger.....see *C and Lang Tools Binder*
dc: desk calculator.....dc(1)
. enable/disable patrol seeks on DCC disks.....sweepon(1M)
consistency check dcheck: filesystem directory.....dcheck(1M)
dd: convert and copy a file.....dd(1M)
ctrace: C program debugger.....see *C and Lang Tools Binder*
pdbx, dbx: parallel debugger.....see *C and Lang Tools Binder*
try to contact remote system with debugging on.....uutry(1M)
crypt: encode/decode.....crypt(1)
timezone: set default system time zone.....timezone(4)
reset the current form field to its default values.....reset(1F)
addusr.def: add user defaults.....addusr_def(4)
format disks or perform other defect management functions.....format(1M)
gtty: set and get terminal state (defunct).....stty(2)
ut_add_user, ut_delete_user: add/delete users in utmp file.....utadduser(3X)
the delta commentary of an SCCS delta.....cdc(1)
delta: make a delta (change) to an SCCS file.....delta(1)
cdc: change the delta commentary of an SCCS delta.....cdc(1)
rmdel: remove a delta from an SCCS file.....rmdel(1)
SCCS file delta: make a delta (change) to an.....delta(1)
comb: combine SCCS deltas.....comb(1)
constructs deroff: remove nroff, tbl, and eqn.....deroff(1)
whatis: describe what a command is.....whatis(1)
close: close a file descriptor.....close(2)
dup: duplicate an open file descriptor.....dup(2)
dup2: duplicate an open file descriptor.....dup2(2)
setmntent: get file system descriptor file entry.....getmntent(3SEQ)
getdtablemax: get maximum in-use descriptor table index.....getdtablemax(2SEQ)
getdtablesize: get descriptor table size.....getdtablesize(2SEQ)
setdtablesize: set descriptor table size.....setdtablesize(2SEQ)
dc: desk calculator.....dc(1)

devbuild, devdestroy: build and destroy virtual devices — devbuild, devdestroy: build and destroy virtual devices ..... devbuild(1M)
devbuild, devdestroy: build and destroy virtual devices ..... devbuild(1M)
ioctl: control device ..... ioctl(2)
prtvtoc: print the VTOC of a block device ..... prtvtoc(1M)
print the VTOC of a standalone disk device ..... prtvtoc(8)
sp: STREAMS pipe device ..... sp(7)
ttydevname: get the terminal associated with a device ..... ttydevname(1)
ttydevname: get the terminal/ device associated with a terminal or ..... ttydevname(1)
diskinfo: disk device characteristic database ..... diskinfo(4)
devnm: device name ..... devnm(1M)
clone: open any minor device on a STREAMS driver ..... clone(7)
devtab: virtual device table ..... devtab(4)
dump kernel auto-configuration device table ..... dumpconf(1M)
backup.dev: backup devices ..... backup_dev(4)
build and destroy virtual devices ..... devbuild(1M)
of virtual-to-physical mapping devices ..... pmap(1M)
SA: devices administered by System Administration ..... sa(7)
constab: constructed devices table ..... constab(4)
devnm: device name ..... devnm(1M)
devtab: virtual device table ..... devtab(4)
convert between major/minor numbers/ dev_to_ttyname, ttyname_to_dev: ..... dev_to_ttyname(3C)
blocks and inodes df: report number of free disk ..... df(1M)
dmesg: collect system diagnostic messages to form error log ..... dmesg(1M)
SCED: SCSI/Ethernet/ Diagnostics board ..... sced(7)
terminal line connection dial: establish an out-going ..... dial(3C)
bdiff: big diff ..... bdiff(1)
diff: differential file comparison ..... diff(1)
comparison diff3: 3-way differential file ..... diff3(1)
sdiff: side-by-side difference program ..... sdiff(1)
diff: differential file comparison ..... diff(1)
diff3: 3-way differential file comparison ..... diff3(1)
dir: format of directories ..... dir(4)
dircmp: directory comparison ..... dircmp(1)
dir: format of directories ..... dir(4)
unlink: link and unlink files and directories ..... link(1M)
mkdir: make directories ..... mkdir(1)
rm, rmdir: remove files or directories ..... rm(1)
uucheck: check the uucp directories and permissions file ..... uucheck(1M)
cd: change working directory ..... cd(1)
chdir: change working directory ..... chdir(2)
chroot: change root directory ..... chroot(2)
get pathname of current working directory ..... getcwd(3C)
system level transition files directory ..... init.d(4)
ls: list contents of directory ..... ls(1)
mkdir: make a directory ..... mkdir(2)
mvdir: move a directory ..... mvdir(1M)
rmdir: remove a directory ..... rmdir(2)
uucleanup: uucp spool directory clean-up ..... uucleanup(1M)
dircmp: directory comparison ..... dircmp(1)
dcheck: filesystem directory consistency check ..... dcheck(1M)
system independent/ getdents: read directory entries and put in a file ..... getdents(2)

dirent: file system independent — directory entry ..... dirent(4)
unlink: remove — directory entry ..... unlink(2)
chroot: change root — directory for a command ..... chroot(1M)
pwd: working — directory name ..... pwd(1)
telldir, seekdir, rewinddir,/ — directory: opendir, readdir, ... directory(3X)
seekdir, rewinddir, closedir: — directory operations ..... directory(3X)
mknod: make a — directory, or a special or ordinary file ..... mknod(2)
directory entry — dirent: file system independent ..... dirent(4)
basename, — dirname: deliver portions of pathnames ..... basename(1)
— dis: code disassembler ..... see *C and Lang Tools Binder*
instructions — dis386: disassemble Series 386 ..... dis386(3SEQ)
t_unbind: — disable a transport endpoint ..... t_unbind(3N)
enable, — disable: enable/disable LP printers ..... enable(1)
enable, disable: enable/ — disable LP printers ..... enable(1)
sweepon, sweepoff: enable/ — disable patrol seeks on DCC disks ..... sweepon(1M)
acct: enable or — disable process accounting ..... acct(2)
swapping, or processor migration — disabled ..... noage(1M)
dis386: — disassemble Series 386 instructions ..... dis386(3SEQ)
dis: object code — disassembler ..... see *C and Lang Tools Binder*
t_rcvdis: retrieve information from — disconnect ..... t_rcvdis(3N)
t_snddis: send user-initiated — disconnect request ..... t_snddis(3N)
installs a boot program on a — disk ..... installboot(1M)
diskusg: generate — disk accounting data by user ID ..... diskusg(1M)
df: report number of free — disk blocks and inodes ..... df(1M)
sd: SCSI — disk controller ..... sd(7)
wd: SSM SCSI — disk controller ..... wd(7)
zd: dual-channel — disk controller ..... zd(7)
disk formatter for dual-channel — disk controller ..... zdformat(8)
print the VTOC of a standalone — disk device ..... prtvtoc(8)
diskinfo: — disk device characteristic database ..... diskinfo(4)
CCSformat: standalone embedded SCSI — disk formatter ..... ccsformat(8)
controller zdformat: standalone — disk formatter for dual-channel disk ..... zdformat(8)
getgeombyname: get — disk geometry by name ..... getgeombyname(3SEQ)
quotactl: manipulate — disk quotas ..... quotactl(2SEQ)
du: summarize — disk usage ..... du(1M)
quota: display — disk usage and limits ..... quota(1)
database — diskinfo: disk device characteristic ..... diskinfo(4)
mkvtoc: populate VTOC on — disks ..... mkvtoc(1M)
enable/disable patrol seeks on DCC — disks ..... sweepon(1M)
mgmt functions format: format — disks or perform other defect ..... format(1M)
data by user ID — diskusg: generate disk accounting ..... diskusg(1M)
and/or the working/ indicator: — display application specific alarms ..... indicator(1F)
quota: — display disk usage and limits ..... quota(1)
vi: screen-oriented (visual) — display editor based on ex ..... vi(1)
man: — display on-line man pages ..... man(1)
virtual-to-physical mapping/ pmap: — display or alter parameters of ..... pmap(1M)
prof: — display profile data ..... prof(1)
bootflags: — display/change system boot flags ..... bootflags(1M)
hypot: Euclidean — distance function ..... hypot(3M)
— dltsectbl: refresh security table ..... dltsectbl(1M)
/dbm_error, dbm_clearerr, — dmb_blkflush: database subroutines ..... ndbm(3X)
messages to form error log — dmesg: collect system diagnostic ..... dmesg(1M)

checkeq: check documents for correct eqn syntax.....checkeq(1)
prctmp,/ chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm,.....acctsh(1M)
strtod, atof: convert string to double-precision number.....strtod(3C)
mrand48, jrand48, srand48, seed48,/ drand48, erand48, lrand48, nrand48,.....drand48(3C)
open any minor device on a STREAMS driver.....clone(7)
cmpt: compatibility partitioning driver.....cmpt(7)
cn: pseudoconsole driver.....cn(7)
terminal multiplexor driver.....st(7)
streampty: STREAMS pseudo terminal driver.....streampty(7)
vtoc: VTOC partitioning driver.....vtoc(7)
Xylogics 781 terminal multiplexor driver.....xs(7)
streamtty: Generic STREAMS tty driver interface.....streamtty(7)
consmem, sm, ss: console memory drivers.....consmem(7)
du: summarize disk usage.....du(1M)
zd: dual-channel disk controller.....zd(7)
/standalone disk formatter for dual-channel disk controller.....zdformat(8)
dump: incremental filesystem dump.....dump(1M)
dump: standalone memory dump.....dump(8)
od: octal dump.....od(1)
of an object file dump: dump parts.....see *C and Lang Tools Binder*
format dump, dumpdates: incremental dump.....dump(4)
dumpfs: dump filesystem information.....dumpfs(1M)
dump, dumpdates: incremental dump format.....dump(4)
dump: incremental filesystem dump.....dump(1M)
device table dumpconf: dump kernel auto-configuration.....dumpconf(1M)
savecore: save a core dump of the operating system.....savecore(1M)
dump: standalone memory dump.....dump(8)
auto-configuration device table dumpconf: dump kernel.....dumpconf(1M)
dump, dumpdates: incremental dump format.....dump(4)
dumpfs: dump filesystem information.....dumpfs(1M)
descriptor dup: duplicate an open file.....dup(2)
descriptor dup2: duplicate an open file.....dup2(2)
dup: duplicate an open file descriptor.....dup(2)
dup2: duplicate an open file descriptor.....dup2(2)
ec: forms compiler.....ec(1M)
echo: echo arguments.....echo(1)
echo: echo arguments.....echo(1)
echo: put string on virtual output.....echo(1F)
floating-point number to string ecvt, fcvt, gcvt: convert.....ecvt(3C)
ed, red: text editor.....ed(1)
end, etext, edata: last locations in program.....end(3C)
command edc: run an externally described.....edc(1M)
casual users) edit: text editor (variant of ex for.....edit(1)
edquota: edit user quotas.....edquota(1M)
sact: print current SCCS file editing activity.....sact(1)
ed, red: text editor.....ed(1)
ex: text editor.....ex(1)
sed: stream editor.....sed(1)
vi: screen-oriented (visual) display editor based on ex.....vi(1)
ld: link editor for object files.....see *C and Lang Tools Binder*
a.out: common assembler and link editor output.....a.out(4)
edit: text editor (variant of ex for casual users).....edit(1)

edquota: edit user quotas ..... edquota(1M)
effective user, real group, and /getgid, getegid: get real user, ... getuid(2)
effective group IDs ..... getuid(2)
effective user, real group, and/ ..... getuid(2)
efl: Extended Fortran Language ..... efl(1)
efl files ..... fsplit(1)
fsplit: split f77, ratfor, or using full regular expressions ..... egrep(1)
egrep: search a file for a pattern ..... egrep(1)
CCSformat: standalone embedded SCSI disk formatter ..... ccsformat(8)
printers enable, disable: enable/disable LP ..... enable(1)
acct: enable or disable process accounting ..... acct(2)
enable, disable: enable/disable LP printers ..... enable(1)
sweepon, sweepoff: enable/disable patrol seeks on DCC disks ..... sweepon(1M)
crypt: encode/decode ..... crypt(1)
crypt, setkey, encrypt: generate hashing encryption ..... crypt(3C)
setkey, encrypt: generate hashing encryption ..... crypt(3C)
crypt: password and file encryption functions ..... crypt(3X)
makekey: generate encryption key ..... makekey(1)
program end, etext, edata: last locations in ..... end(3C)
gencc: create a front- end to the cc command ..... gencc(1M)
/getgrgid, getgrnam, setgrent, endgrent, fgetgrent: get group file entry ..... getgrent(3C)
setmntent: get file/ addmntent, endmntent, getmntent, hasmntopt, ... getmntent(3SEQ)
bind an address to a transport endpoint ..... t_bind(3N)
t_close: close a transport endpoint ..... t_close(3N)
at the current event on a transport endpoint ..... t_look(3N)
t_open: establish a transport endpoint ..... t_open(3N)
manage options for a transport endpoint ..... t_optmgmt(3N)
t_unbind: disable a transport endpoint ..... t_unbind(3N)
/getpwuid, getpwnam, setpwent, endpwent, fgetpwent: get password file/ ..... getpwent(3C)
getspent, getspnam, setspent, endspent, fgetspent, lckpwdf,/ ..... getspent(3X)
/getutline, pututline, setutent, endutent, utmpname, getutpid: access/ ..... getut(3C)
mailbug: submit a bug report or enhancement request ..... mailbug(1)
execution env: set environment for command ..... env(1)
files .environ, .pref, .variables: FACE ..... env(4)
environ: user environment ..... environ(5)
environ: user environment ..... environ(5)
putenv: change or add value to environment ..... putenv(3C)
commands performed for multi-user environment ..... rc2(1M)
profile: setting up an environment at login time ..... profile(4)
fpsetsticky: IEEE floating-point environment control ..... fpgetround(3C)
env: set environment for command execution ..... env(1)
getenv: return value for environment name ..... getenv(3C)
set, unset: set and unset environment variables in core or in files ..... set(1F)
deroff: remove nroff, tbl, and eqn constructs ..... deroff(1)
checkeq: check documents for correct eqn syntax ..... checkeq(1)
jrand48, srand48, seed48,/ drand48, erand48, lrand48, nrand48, mrand48, ... drand48(3C)
complementary error function erf, erfc: error function and ..... erf(3M)
complementary error function erf, erfc: error function and ..... erf(3M)
error messages perror, errno, sys_errlist, sys_nerr: system ..... perror(3C)
error function and complementary error function ..... erf(3M)
error function erf, erfc: error function and complementary ..... erf(3M)
t_rcvuderr: receive a unit data error indication ..... t_rcvuderr(3N)
system diagnostic messages to form error log ..... dmesg(1M)
errorlog: internal error log file ..... errorlog(4)

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)
t_error: produce error message ..... t_error(3N)
errormsg: error message text file ..... errormsg(4)
errno, sys_errlist, sys_nerr: system error messages ..... perror(3C)
introduction to system calls and error numbers ..... intro(2)
matherr: error-handling function ..... matherr(3M)
errorlog: internal error log file ..... errorlog(4)
errormsg: error message text file ..... errormsg(4)
spellin, hashcheck: find spelling errors ..... spell(1)
/etc/shadow with information from / etc/passwd ..... pwconv(1M)
pwconv: installs and updates / etc/shadow with information from/ ..... pwconv(1M)
end, etext, edata: last locations in program ..... end(3C)
SCED: SCSI/ Ethernet/Diagnostics board ..... sced(7)
hypot: Euclidean distance function ..... hypot(3M)
expr: evaluate arguments as an expression ..... expr(1)
test: condition evaluation command ..... test(1)
t_look: look at the current event on a transport endpoint ..... t_look(3N)
to STREAMS error logging and event tracing ..... log(7)
sulog: log of substitute user events ..... sulog(4)
(visual) display editor based on ex ..... vi(1)
edit: text editor (variant of ex for casual users) ..... edit(1)
ex: text editor ..... ex(1)
execlp, execvp: execute a file exec: execl, execv, execle, execve, ... exec(2)
execlp, execvp: execute a/ exec: execl, execv, execle, execve, ... exec(2)
exec: execl, execv, execle, execve, execle, execve, execlp, execvp: execute a file ..... exec(2)
exec: execl, execv, execle, execve, execlp, execvp: execute a file ..... exec(2)
run: run an executable ..... run(1F)
execle, execve, execlp, execvp: execute a file ..... exec(2)
m_fork: execute a subprogram in parallel ..... m_fork(3PPS)
construct argument list(s) and execute command ..... xargs(1)
at, batch: execute commands at a later time ..... at(1)
regcmp, regex: compile and execute regular expression ..... regcmp(3X)
uuxqt: execute remote command requests ..... uuxqt(1M)
env: set environment for command execution ..... env(1)
suspend and resume child process execution ..... m_park_procs(3PPS)
/system command execution ..... uux(1C)
sleep: suspend execution for an interval ..... sleep(1)
sleep: suspend execution for interval ..... sleep(3C)
monitor: prepare execution profile ..... monitor(3C)
profil: execution time profile ..... profil(2)
execvp: execute a file exec: execl, execv, execle, execve, execlp, ... exec(2)
exec: execl, execv, execle, execve, execlp, execvp: execute a file ..... exec(2)
execv, execle, execve, execlp, execvp: execute a file ..... exec(2)
exit, _exit: terminate process ..... exit(2)
_exit: terminate process ..... exit(2)
exponential, logarithm, power/ exp, log, log10, pow, sqrt: ..... exp(3M)
uncompress, zcat: compress and expand data ..... compress(1)
pack, pcat, unpack: compress and expand files ..... pack(1)
exp, log, log10, pow, sqrt: exponential, logarithm, power, square/ ..... exp(3M)
expression expr: evaluate arguments as an ..... expr(1)

expr: evaluate arguments as an expression ..... expr(1)
regex: compile and execute regular expression ..... regcmp(3X)
regcmp: regular expression compile ..... regcmp(1)
regexp: regular expression compile and match routines ..... regexp(5)
for a pattern using full regular expressions ..... egrep(1)
efl: Extended Fortran Language ..... efl(1)
edc: run an externally described command ..... edc(1M)
fsplit: split f77, ratfor, or efl files ..... fsplit(1)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value/ ..... floor(3M)
pathalias: alias file for FACE ..... alias(4)
.environ, .pref, .variables: FACE files ..... env(4)
report inter-process communication facilities status ..... ipcs(1)
a number factor: obtain the prime factors of ..... factor(1)
factor: obtain the prime factors of a number ..... factor(1)
true, false: provide truth values ..... true(1)
abort: generate an IOT fault ..... abort(3C)
stream fclose, fflush: close or flush a ..... fclose(3S)
fcntl: file control ..... fcntl(2)
fcntl: file control options ..... fcntl(5)
number to string ecvt, fcvt, gcvt: convert floating-point ..... ecvt(3C)
fopen, freopen, fdopen: open a stream ..... fopen(3S)
ferror, feof, clearerr, fileno: stream status inquiries ..... ferror(3S)
stream status inquiries ferror, feof, clearerr, fileno: ..... ferror(3S)
for a filesystem ff: list filenames and statistics ..... ff(1M)
fclose, fflush: close or flush a stream ..... fclose(3S)
from a stream getc, getchar, fgetc, getw: get character or word ..... getc(3S)
/getgrnam, setgrent, endgrent, fgetgrent: get group file entry ..... getgrent(3C)
/getpwnam, setpwent, endpwent, fgetpwent: get password file entry ..... getpwent(3C)
gets, fgets: get a string from a stream ..... gets(3S)
/getspnam, setspent, endspent, fgetspent, lckpwdf, ulckpwdf: get/ ..... getspent(3X)
string fgrep: search a file for a character ..... fgrep(1)
cut: cut out selected fields of each line of a file ..... cut(1)
mkfifo: make a FIFO special file ..... mkfifo(3C)
access: determine accessibility of a file ..... access(2)
chmod: change mode of file ..... chmod(2)
chown: change owner and group of a file ..... chown(2)
core: format of core image file ..... core(4)
cronlog: cron history file ..... cronlog(4)
crontab: user crontab file ..... crontab(1)
crontabs: clock daemon file ..... crontabs(4)
ctags: create a tags file ..... ctags(1)
selected fields of each line of a file ..... cut(1)
dd: convert and copy a file ..... dd(1M)
make a delta (change) to an SCCS file ..... delta(1)
errorlog: internal error log file ..... errorlog(4)
errormsg: error message text file ..... errormsg(4)
execve, execlp, execvp: execute a file ..... exec(2)
get: get a version of an SCCS file ..... get(1)
group: group file ..... group(4)
issue: issue identification file ..... issue(4)
header of a member of an archive file ..... ldahread(3X)
ldaclose: close a common object file ..... ldclose(3X)

the file header of a common object file.....ldfhread(3X)
of a section of a common object file.....ldlseek(3X)
file header of a common object file.....ldohseek(3X)
of a section of a common object file.....ldrseek(3X)
section header of a common object file.....ldshread(3X)
section of a common object file.....ldsseek(3X)
table entry of a common object file.....ldtbindex(3X)
table entry of a common object file.....ldtbread(3X)
the symbol table of a common object file.....ldtbseek(3X)
number entries in a common object file.....linenum(4)
link: link to a file.....link(2)
mkfifo: make a FIFO special file.....mkfifo(3C)
mknod: build special file.....mknod(1M)
directory, or a special or ordinary file.....mknod(2)
format of a module configuration file.....mod_config(4)
motd: message-of-the-day file.....motd(4)
newform: change the format of a text file.....newform(1)
null: the null file.....null(7)
passwd: password file.....passwd(4)
files or subsequent lines of one file.....paste(1)
product: installed products file.....product(4)
prs: print an SCCS file.....prs(1)
rcsfile: format of RCS file.....rcsfile(4)
read: read from file.....read(2)
readv: read vector from file.....readv(2SEQ)
reinit: run an initialization file.....reinit(1F)
information for a common object file.....reloc(4)
rename: change the name of a file.....rename(2)
rmdel: remove a delta from an SCCS file.....rmdel(1)
compare two versions of an SCCS file.....sccsdiff(1)
sccsfile: format of SCCS file.....sccsfile(4)
sccstorcs: build RCS file from SCCS file.....sccstorcs(1)
section header for a common object file.....scnhdr(4)
format of curses screen image file.....scr_dump(4)
shadow: restricted password file.....shadow(4)
system shutdown information file.....shutdown(4)
spellhist: unmatched spelling words file.....spellhist(4)
print checksum and block count of a file.....sum(1)
symlink: make symbolic link to a file.....symlink(2SEQ)
format of a system configuration file.....sys_config(4)
tail: deliver the last part of a file.....tail(1)
term: format of compiled term file.....term(4)
tmpfile: create a temporary file.....tmpfile(3S)
create a name for a temporary file.....tmpnam(3S)
access and modification times of a file.....touch(1)
undo a previous get of an SCCS file.....unget(1)
uniq: report repeated lines in a file.....uniq(1)
add/delete users in utmp file.....utadduser(3X)
the uucp directories and permissions file.....uucheck(1M)
val: validate SCCS file.....val(1)
write: write on a file.....write(2)
writev: write vector on a file.....writev(2SEQ)

utime: set file access and modification times.....utime(2)
ldfcn: common object file access routines.....ldfcn(4)
readfile, longline: reads file and gets longest line.....readfile(1F)
cpset: copy file and set mode, user, and group IDs.....cpset(1)
Hewlett-Packard 2645A terminal tape file archiver.....hpio(1)
tar: tape file archiver.....tar(1)
cpio: copy file archives in and out.....cpio(1)
rcs: change RCS file attributes.....rcs(1)
pwck, grpck: password/group file checkers.....pwck(1M)
mcs: manipulate the object file comment section.....mcs(1)
diff: differential file comparison.....diff(1)
diff3: 3-way differential file comparison.....diff3(1)
fcntl: file control.....fcntl(2)
fcntl: file control options.....fcntl(5)
UNIX-to-UNIX system file copy.....uuto(1C)
umask: set and get file creation mask.....umask(2)
close: close a file descriptor.....close(2)
dup: duplicate an open file descriptor.....dup(2)
dup2: duplicate an open file descriptor.....dup2(2)
file: determine file type.....file(1)
sact: print current SCCS file editing activity.....sact(1)
crypt: password and file encryption functions.....crypt(3X)
endgrent, fgetgrent: get group file entry.....getgrent(3C)
get file system descriptor file entry.....getmntent(3SEQ)
endpwent, fgetpwent: get password file entry.....getpwent(3C)
ulckpwdf: get shadow password file entry.....getspent(3X)
utmpname, getutpid: access utmp file entry.....getut(3C)
putpwent: write password file entry.....putpwent(3C)
putspent: write shadow password file entry.....putspent(3X)
fgrep: search a file for a character string.....fgrep(1)
grep: search a file for a pattern.....grep(1)
egrep: search a file for a pattern using full regular expressions.....egrep(1)
proto: prototype job file for at.....proto(4)
pathalias: alias file for FACE.....alias(4)
ldaopen: open a common object file for reading.....ldopen(3X)
acct: per-process accounting file format.....acct(4)
ar: common archive file format.....see *C and Lang Tools Binder*
manager, updates text files in ISIS file format.....isismgr(1M)
intro: introduction to file formats.....intro(4)
sccstorcs: build RCS file from SCCS file.....sccstorcs(1)
number entries of a common object file function.....ldlread(3X)
filehdr: file header for common object files.....filehdr(4)
limits: file header for implementation-specific/.....limits(4)
unistd: file header for symbolic constants.....unistd(4)
ldfhread: read the file header of a common object file.....ldfhread(3X)
ldohseek: seek to the optional file header of a common object file.....ldohseek(3X)
split: split a file into pieces.....split(1)
mmap: map an open file into the processs address space.....mmap(2SEQ)
ISIS file format isismgr: ISIS file manager, updates text files in.....isismgr(1M)
merge: three-way file merge.....merge(1)
ttyslot: find the slot in the utmp file of the current user.....ttyslot(3C)
more: view a file one full screen at a time.....more(1)

fuser: identify processes using a file or file structure.....fuser(1M)
creat: create a new file or rewrite an existing one.....creat(2)
pg: file perusal filter for CRTs.....pg(1)
lseek: move read/write file pointer.....lseek(2)
fseek, rewind, ftell: reposition a file pointer in a stream.....fseek(3S)
bfs: big file scanner.....bfs(1)
from a two to a one password file scheme.....pwunconv(1M)
stat, lstat, fstat: get file status.....stat(2)
identify processes using a file or file structure.....fuser(1M)
symbol name for common object file symbol table entry.....ldgetname(3X)
syms: common object file symbol table format.....syms(4)
newfs: construct a new file system.....newfs(1M)
/read directory entries and put in a file system independent format.....getdents(2)
/getmntent, hasmntopt, setmntent: get file system descriptor file entry.....getmntent(3SEQ)
dirent: file system independent directory entry.....dirent(4)
fs, inode: format of file system volume.....fs(4)
truncate, ftruncate: truncate a file to a specified length.....truncate(2SEQ)
uusched: the scheduler for the uucp file transport program.....uusched(1M)
uucico: file transport program for the uucp system.....uucico(1M)
ftw: walk a file tree.....ftw(3C)
file: determine file type.....file(1)
umask: set file-creation mode mask.....umask(1)
object files filehdr: file header for common.....filehdr(4)
mktemp: make a unique filename.....mktemp(3C)
search Interpreter criteria for filename.....pathconv(1F)
ctermid: generate filename for terminal.....ctermid(3S)
ff: list filenames and statistics for a filesystem.....ff(1M)
ferror, feof, clearerr, fileno: stream status inquiries.....ferror(3S)
search and print process accounting files.....acctcom(1)
merge or add total accounting files.....acctmerg(1M)
admin: create and administer SCCS files.....admin(1)
cat: concatenate and print files.....cat(1)
cmp: compare two files.....cmp(1)
or reject lines common to two sorted files.....comm(1)
config: build system configuration files.....config(1M)
cp, ln, mv: copy, link or move files.....cp(1)
.environ, .pref, .variables: FACE files.....env(4)
file header for common object files.....filehdr(4)
find: find files.....find(1)
fspec: format specification in text files.....fspec(4)
fsplit: split f77, ratfor, or efl files.....fsplit(1)
string, format of graphical files.....gps(4)
ident: identify files.....ident(1)
intro: introduction to special files.....intro(7)
lockf: record locking on files.....lockf(3C)
makedev: make system special files.....makedev(1M)
pcat, unpack: compress and expand files.....pack(1)
pr: print files.....pr(1)
and other information about RCS files.....rlog(1)
environment variables in core or in files.....set(1F)
sort: sort and/or merge files.....sort(1)
uucp: networking support files.....uucp(4)

what: identify SCCS files ..... what(1)
link, unlink: link and unlink files and directories ..... link(1M)
init.d: system level transition files directory ..... init.d(4)
ISIS file manager, updates text files in ISIS file format ..... isismgr(1M)
passmgmt: password files management ..... passmgmt(1M)
rm, rmdir: remove files or directories ..... rm(1)
paste: merge same lines of several files or subsequent lines of one/ ..... paste(1)
ott: files that hold object architecture information ..... ott(4)
checkfsys: check a filesystem ..... checkfsys(1M)
list filenames and statistics for a filesystem ..... ff(1M)
mkfs: construct a filesystem ..... mkfs(1M)
mount: mount a filesystem ..... mount(2)
repquota: summarize quotas for a filesystem ..... repquota(1M)
tunefs: tune up an existing filesystem ..... tunefs(1M)
umount: unmount a filesystem ..... umount(2)
volcopy: make literal copy of filesystem ..... volcopy(1M)
ckbupscd: check filesystem backup schedule ..... ckbupscd(1M)
interactive repair fsck: filesystem consistency check and ..... fsck(1M)
dcheck: filesystem directory consistency check ..... dcheck(1M)
dump: incremental filesystem dump ..... dump(1M)
fstyp: determine filesystem identifier ..... fstyp(1M)
dumpfs: dump filesystem information ..... dumpfs(1M)
statfs, fstatfs: get filesystem information ..... statfs(2)
quotacheck: filesystem quota consistency checker ..... quotacheck(1M)
quotaon, quotaoff: turn filesystem quotas on and off ..... quotaon(1M)
restore: incremental filesystem restore ..... restore(1M)
ustat: get filesystem statistics ..... ustat(2)
fsstat: report filesystem status ..... fsstat(1M)
icheck: filesystem storage consistency check ..... icheck(1M)
mnttab: mounted filesystem table ..... mnttab(4)
labelit: provide labels for filesystems ..... labelit(1M)
mount, umount: mount and unmount filesystems ..... mount(1M)
umountall: mount, unmount multiple filesystems ..... mountall(1M)
checklist: list of filesystems processed by fsck and ncheck ..... checklist(4)
fstab: file-system-table ..... fstab(4)
greek: select terminal filter ..... greek(1)
nl: line numbering filter ..... nl(1)
pg: file perusal filter for CRTs ..... pg(1)
col: filter reverse line-feeds ..... col(1)
lpfilter: administer filters used with the LP print service ..... lpfilter(1M)
find: find files ..... find(1)
for an object library lorder: find ordering relation ..... see *C and Lang Tools Binder*
display/change system boot flags ..... bootflags(1M)
/fpgetsticky, fpsetsticky: IEEE floating-point environment control ..... fpgetround(3C)
isnan: isnand, isnanf: test for floating-point NaN (Not-A-Number) ..... isnan(3C)
ecvt, fcvt, gcvt: convert floating-point number to string ..... ecvt(3C)
ldexp, modf: manipulate parts of floating-point numbers ..... frexp(3C)
ceiling, remainder, absolute value/ floor, ceil, fmod, fabs: floor, ... floor(3M)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value/ ..... floor(3M)
fclose, fflush: close or flush a stream ..... fclose(3S)
a co-process with the controlling FMLI object ..... vsig(1F)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute/ ..... floor(3M)

stream fopen, freopen, fdopen: open a ..... fopen(3S)
tcgetpgrp: get foreground process group ID..... tcgetpgrp(2)
tcsetpgrp: set foreground process group ID..... tcsetpgrp(2)
fork: create a new process..... fork(2)
system diagnostic messages to form error log..... dmesg(1M)
reset: reset the current form field to its default values ..... reset(1F)
acct: per-process accounting file format ..... acct(4)
dump, dumpdates: incremental dump format..... dump(4)
put in a file system independent format..... getdents(2)
updates text files in ISIS file format..... isismgr(1M)
common object file symbol table format..... syms(4)
management functions format: format disks or perform other defect..... format(1M)
other defect management functions format: format disks or perform ..... format(1M)
mod_config: format of a module configuration file..... mod_config(4)
sys_config: format of a system configuration file..... sys_config(4)
newform: change the format of a text file..... newform(1)
term: format of compiled term file..... term(4)
core: format of core image file..... core(4)
cpio: format of cpio archive..... cpio(4)
scr_dump: format of curses screen image file..... scr_dump(4)
dir: format of directories..... dir(4)
fs, inode: format of file system volume ..... fs(4)
gps: graphical primitive string, format of graphical files..... gps(4)
rcsfile: format of RCS file..... rcsfile(4)
sccsfile: format of SCCS file..... sccsfile(4)
fspec: format specification in text files..... fspec(4)
strftime: format time information ..... strftime(3C)
intro: introduction to file formats..... intro(4)
utmp, wtmp: utmp and wtmp entry formats..... utmp(4)
scanf, fscanf, sscanf: convert formatted input ..... scanf(3S)
printf, fprintf, sprintf: print formatted output ..... printf(3S)
vprintf, vfprintf, vsprintf: print formatted output of a varargs/..... vprintf(3S)
pwfix: remove incorrectly formatted password entries..... pwfix(1M)
standalone embedded SCSI disk formatter..... ccsformat(8)
zdformat: standalone disk formatter for dual-channel disk/..... zdformat(8)
ASSIST menus or command forms..... astgen(1)
ec: forms compiler ..... ec(1M)
lpforms: administer forms used with the LP print service..... lpforms(1M)
addformopt: add a form/SELECT window option to a menu ..... addformopt(1M)
ratfor: rational Fortran dialect..... ratfor(1)
efl: Extended Fortran Language ..... efl(1)
variables pathconf, fpathconf: get configurable pathname..... pathconf(2)
fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky/..... fpgetround(3C)
fpsetmask, fpgetsticky/ fpgetround, fpsetround, fpgetmask, ... fpgetround(3C)
/fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky: IEEE/..... fpgetround(3C)
printf, fprintf, sprintf: print formatted output..... printf(3S)
fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky:/ ..... fpgetround(3C)
fpgetsticky,/ fpgetround, fpsetround, fpgetmask, fpsetmask, ... fpgetround(3C)
/fpgetmask, fpsetmask, fpgetsticky, fpsetsticky: IEEE floating-point/..... fpgetround(3C)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
puts, fputs: put a string on a stream..... puts(3S)
getfrm: returns the current frame number ..... getfrm(1F)

fread, fwrite: binary input/output ..... fread(3S)
t_free: free a library structure ..... t_free(3N)
df: report number of free disk blocks and inodes ..... df(1M)
malloc, free, realloc, calloc: main memory allocator ..... malloc(3C)
mallinfo: fast main memory/ malloc, free, realloc, calloc, mallopt, ... malloc(3X)
fopen, freopen, fdopen: open a stream ..... fopen(3S)
of floating-point numbers frexp, ldexp, modf: manipulate parts ..... frexp(3C)
gencc: create a front-end to the cc command ..... gencc(1M)
volume fs, inode: format of file system ..... fs(4)
scanf, fscanf, sscanf: convert formatted input ..... scanf(3S)
list of filesystems processed by fsck and ncheck ..... checklist(4)
and interactive repair fsck: filesystem consistency check ..... fsck(1M)
file pointer in a stream fseek, rewind, ftell: reposition a ..... fseek(3S)
generation numbers fsirand: install random inode ..... fsirand(1M)
files fspec: format specification in text ..... fspec(4)
files fsplit: split f77, ratfor, or efl ..... fsplit(1)
fsstat: report filesystem status ..... fsstat(1M)
fstab: file-system-table ..... fstab(4)
stat, lstat, fstat: get file status ..... stat(2)
statfs, fstatfs: get filesystem information ..... statfs(2)
identifier fstyp: determine filesystem ..... fstyp(1M)
fseek, rewind, ftell: reposition a file pointer in a stream ..... fseek(3S)
communication package stdipc: ftok: standard interprocess ..... stdipc(3C)
specified length truncate, ftruncate: truncate a file to a ..... truncate(2SEQ)
ftw: walk a file tree ..... ftw(3C)
function and complementary error function ..... erf(3M)
gamma: log gamma function ..... gamma(3M)
hypot: Euclidean distance function ..... hypot(3M)
entries of a common object file function ..... ldlread(3X)
matherr: error-handling function ..... matherr(3M)
prof: profile within a function ..... prof(5)
erf, erfc: error function and complementary error function ..... erf(3M)
j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
crypt: password and file encryption functions ..... crypt(3X)
logarithm, power, square root functions ..... exp(3M)
ceiling, remainder, absolute value functions ..... floor(3M)
or perform other defect management functions ..... format(1M)
sinh, cosh, tanh: hyperbolic functions ..... sinh(3M)
sysi86: machine specific functions ..... sysi86(2)
keys to internal menu system functions ..... syskeymap(4)
acos, atan, atan2: trigonometric functions ..... trig(3M)
math: math functions and constants ..... math(5)
intro: introduction to functions and libraries ..... intro(3)
300, 300s: handle special functions of DASI 300 and 300s terminals ..... 300(1)
hp: handle special functions of Hewlett-Packard terminals ..... hp(1)
450: handle special functions of the DASI 450 terminal ..... 450(1)
file or file structure fuser: identify processes using a ..... fuser(1M)
specified controlling terminal fvhangup: virtually hangup the ..... fvhangup(2SEQ)
fread, fwrite: binary input/output ..... fread(3S)
accounting records fwtmp, wtmpfix: manipulate connect ..... fwtmp(1M)
gamma: log gamma function ..... gamma(3M)
gamma: log gamma function ..... gamma(3M)

ecvt, fcvt, gcvt: convert floating-point number to string .....ecvt(3C)
command gencc: create a front-end to the cc .....gencc(1M)
command forms astgen: program for generating/modifying ASSIST menus or .....astgen(1)
fsirand: install random inode generation numbers .....fsirand(1M)
rand, srand: simple random-number generator .....rand(3C)
streamtty: Generic STREAMS tty driver interface .....streamtty(7)
tty: controlling terminal (gentty) interface .....tty(7)
getgeombyname: get disk geometry by name .....getgeombyname(3SEQ)
character or word from a stream getc, getchar, fgetc, getw: get .....getc(3S)
or word from a stream getc, getchar, fgetc, getw: get character .....getc(3S)
working directory getcwd: get pathname of current .....getcwd(3C)
put in a file system independent/ getdents: read directory entries and .....getdents(2)
descriptor table index getdtablemax: get max in-use .....getdtablemax(2SEQ)
size getdtablesize: get descriptor table .....getdtablesize(2SEQ)
getuid, geteuid, getgid, getegid: get real user, effective user, real/ .....getuid(2)
name getenv: return value for environment .....getenv(3C)
user, effective user, real/ getuid, geteuid, getgid, getegid: get real .....getuid(2)
number getfrm: returns the current frame .....getfrm(1F)
geometry by name getgeombyname: get disk/ .....getgeombyname(3SEQ)
getuid, geteuid, getgid, getegid: get real user, effective user/ .....getuid(2)
setgrent, endgrent, fgetgrent: get/ getgrent, getgrgid, getgrnam, ...getgrent(3C)
endgrent, fgetgrent: get/ getgrent, getgrgid, getgrnam, setgrent, ...getgrent(3C)
fgetgrent: get/ getgrent, getgrgid, getgrnam, setgrent, endgrent, ...getgrent(3C)
IDs getgroups: get supplementary group .....getgroups(2)
currently marked menu items getitems: returns a list of the .....getitems(1F)
getlogin: get login name .....getlogin(3C)
file system/ addmntent, endmntent, getmntent, hasmntopt, setmntent: get ....getmntent(3SEQ)
stream getmsg: get next message off a .....getmsg(2)
argument vector getopt: get option letter from .....getopt(3C)
getopt: parse command options .....getopt(1)
getopts, getoptcvt: parse command options .....getopts(1)
options getopts, getoptcvt: parse command .....getopts(1)
getpagesize: get system page size .....getpagesize(2SEQ)
getpass: read a password .....getpass(3C)
process group, and parent/ getpid, getpgrp, getppid: get process, ...getpid(2)
process, process group, and parent/ getpid, getpgrp, getppid: get .....getpid(2)
and parent process/ getpid, getpgrp, getppid: get process, process group, ...getpid(2)
utilization information get_process_stats: get resource ....get_process_stats(2SEQ)
getpseudotty: get pseudo terminals .....getpseudotty(3SEQ)
getpw: get name from IUD .....getpw(3C)
setpwent, endpwent, fgetpwent: get/ getpwent, getpwuid, getpwnam, ...getpwent(3C)
fgetpwent: get/ getpwent, getpwuid, getpwnam, setpwent, endpwent, ...getpwent(3C)
endpwent, fgetpwent: get/ getpwent, getpwuid, getpwnam, setpwent, ...getpwent(3C)
stream gets, fgets: get a string from a .....gets(3S)
readfile, longline: reads file and gets longest line .....readfile(1F)
scsiinfo structure getscsiinfo, getscsimatch: get .....getscsiinfo(3SEQ)
getscsiinfo, getscsimatch: get scsiinfo structure .....getscsiinfo(3SEQ)
endspent, fgetspent, lckpwdf,/ getspent, getspnam, setspent, ...getspent(3X)
fgetspent, lckpwdf,/ getspent, getspnam, setspent, endspent, ...getspent(3X)
speed and terminal settings used by getty .....gettydefs(4)
speed, and line discipline getty: set terminal type, modes, ...getty(1M)
ct: spawn getty to a remote terminal .....ct(1C)

settings used by getty — gettydefs: speed and terminal ..... gettydefs(4)
get real user, effective user, real/ — getuid, geteuid, getgid, getegid: ..... getuid(2)
microsecond clock — getusclk, usclk_init: get/init ..... getusclk(3SEQ)
pututline, setutent, endutent,/ — getut: getutent, getutid, getutline, ... getut(3C)
pututline, setutent,/ getut: — getutent, getutid, getutline, ... getut(3C)
getut: getutent, — getutid, getutline, pututline, setutent,/ ..... getut(3C)
getut: getutent, getutid, — getutline, pututline, setutent, endutent,/ ..... getut(3C)
/setutent, endutent, utmpname, — getutpid: access utmp file entry ..... getut(3C)
getc, getchar, fgetc, — getw: get character or word from a stream ..... getc(3S)
zdinfo structure — getzdinfo, getzdinfobyname: get ..... getzdinfo(3SEQ)
getzdinfo, — getzdinfobyname: get zdinfo structure ..... getzdinfo(3SEQ)
m_next: increment — global counter ..... m_next(3PPS)
task: — global security table ..... task(4)
operating system terms and symbols — glossary: definitions of common ..... glossary(1)
mktime, tzset:/ ctime, localtime, — gmtime, asctime, cftime, ascftime, ... ctime(3C)
setjmp, longjmp: non-local — goto ..... setjmp(3C)
format of graphical files — gps: graphical primitive string, ... gps(4)
primitive string, format of — graphical files ..... gps(4)
of graphical files gps: — graphical primitive string, format ..... gps(4)
plot: — graphics interface ..... plot(4)
plot: — graphics interface subroutines ..... plot(3X)
— greek: select terminal filter ..... greek(1)
— grep: search a file for a pattern ..... grep(1)
chown, chgrp: change owner or — group ..... chown(1)
newgrp: log in to a new — group ..... newgrp(1M)
run command in its own process — group ..... setpgrp(1)
initgroups: initialize — group access list ..... initgroups(3SEQ)
setgroups: set — group access list ..... setgroups(2SEQ)
/get real user, effective user, real — group, and effective group IDs ..... getuid(2)
/getppid: get process, process — group, and parent process IDs ..... getpid(2)
group: — group file ..... group(4)
pwck, grpck: password/ — group file checkers ..... pwck(1M)
setgrent, endgrent, fgetgrent: get — group file entry ..... getgrent(3C)
— group: group file ..... group(4)
setpgrp: set process — group ID ..... setpgrp(2)
create session and set process — group ID ..... setsid(2)
tcgetpgrp: get foreground process — group ID ..... tcgetpgrp(2)
tcsetpgrp: set foreground process — group ID ..... tcsetpgrp(2)
copy file and set mode, user, and — group IDs ..... cpset(1)
getgroups: get supplementary — group IDs ..... getgroups(2)
user, real group, and effective — group IDs ..... getuid(2)
setuid, setgid: set user and — group IDs ..... setuid(2)
id: print user and — group IDs and names ..... id(1M)
groups: show — group memberships ..... groups(1)
chown: change owner and — group of a file ..... chown(2)
send a signal to a process or a — group of processes ..... kill(2)
maintain, update, and regenerate — groups of programs ..... make(1)
— groups: show group memberships ..... groups(1)
pwck, — grpck: password/group file checkers ..... pwck(1M)
ssignal, — gsignal: software signals ..... ssignal(3C)
stty, — gtty: set and get terminal state (defunct) ..... stty(2)
terminal fvhangup: virtually — hangup the specified controlling ..... fvhangup(2SEQ)

nohup: run a command immune to ... hangups and quits.....nohup(1)
hsearch, hcreate, hdestroy: manage ... hash search tables.....hsearch(3C)
spell, hashmake, spellin, ... hashcheck: find spelling errors .....spell(1)
crypt, setkey, encrypt: generate ... hashing encryption .....crypt(3C)
spelling errors spell, ... hashmake, spellin, hashcheck: find.....spell(1)
addmntent, endmntent, getmntent, ... hasmntopt, setmntent: get file/.....getmntent(3SEQ)
search tables hsearch, ... hcreate, hdestroy: manage hash.....hsearch(3C)
hsearch, hcreate, ... hdestroy: manage hash search tables .....hsearch(3C)
scnhdr: section ... header for a common object file.....scnhdr(4)
filehdr: file ... header for common object files.....filehdr(4)
limits: file ... header for implementation-specific constants.....limits(4)
unistd: file ... header for symbolic constants.....unistd(4)
ldfhread: read the file ... header of a common object file.....ldfhread(3X)
ldohseek: seek to the optional file ... header of a common object file .....ldohseek(3X)
/read an indexednamed section ... header of a common object file.....ldshread(3X)
ldahread: read the archive ... header of a member of an archive file.....ldahread(3X)
helplog: monitor ... help actions.....helplog(4)
help: operating system ... Help Facility.....help(1)
helpadm: change the ... Help Facility database .....helpadm(1M)
... help: operating system Help Facility.....help(1)
database ... helpadm: change the Help Facility .....helpadm(1M)
... helplog: monitor help actions.....helplog(4)
file archiver hpio: ... Hewlett-Packard 2645A terminal tape .....hpio(1)
hp: handle special functions of ... Hewlett-Packard terminals .....hp(1)
cronlog: cron ... history file.....cronlog(4)
ott: files that ... hold object architecture information.....ott(4)
uptime: show ... how long system has been up.....uptime(1)
Hewlett-Packard terminals ... hp: handle special functions of.....hp(1)
tape file archiver ... hpio: Hewlett-Packard 2645A terminal .....hpio(1)
hash search tables ... hsearch, hcreate, hdestroy: manage.....hsearch(3C)
sinh, cosh, tanh: ... hyperbolic functions .....sinh(3M)
... hypot: Euclidean distance function .....hypot(3M)
u3b5, vax: get processor/ machid: ... i386, ns32000, pdp11, u3b, u3b2,.....machid(1)
consistency check ... icheck: filesystem storage.....icheck(1M)
... ident: identify files.....ident(1)
m_get_myid: return process ... identification.....m_get_myid(3PPS)
issue: issue ... identification file .....issue(4)
fstyp: determine filesystem ... identifier .....fstyp(1M)
shmget: get shared memory segment ... identifier.....shmget(2)
file and set mode, user, and group ... IDs.....cpset(1)
getgroups: get supplementary group ... IDs.....getgroups(2)
process group, and parent process ... IDs.....getpid(2)
real group, and effective group ... IDs.....getuid(2)
setuid, setgid: set user and group ... IDs.....setuid(2)
id: print user and group ... IDs and names .....id(1M)
/fpsetmask, fpgetsticky, fpsetsticky: ... IEEE floating-point environment/.....fpgetround(3C)
core: format of core ... image file.....core(4)
scr_dump: format of curses screen ... image file .....scr_dump(4)
crash: examine system ... images.....crash(1M)
limits: file header for ... implementation-specific constants .....limits(4)
pwfix: remove ... incorrectly formatted password entries.....pwfix(1M)
m_next: ... increment global counter.....m_next(3PPS)

dump, dumpdates: incremental dump format.....dump(4)
dump: incremental filesystem dump.....dump(1M)
restore: incremental filesystem restore.....restore(1M)
get maximum in-use descriptor table index.....getdtablemax(2SEQ)
common/ ldtbindex: compute the index of a symbol table entry of a.....ldtbindex(3X)
common object/ ldtbread: read an indexed symbol table entry of a.....ldtbread(3X)
common/ldshread, ldnshread: read an indexednamed section header of a.....ldshread(3X)
ldsseek, ldnsseek: seek to an indexednamed section of a common/.....ldsseek(3X)
receipt of an orderly release indication.....t_rcvrel(3N)
receive a unit data error indication.....t_rcvuderr(3N)
specific alarms and/or the working indicator.....indicator(1F)
specific alarms and/or the working/ indicator: display application.....indicator(1F)
calculate number of blocks and indirect blocks.....nblocks(3SEQ)
syscall: indirect system call.....syscall(2)
terminfo descriptions infocmp: compare or print out.....infocmp(1M)
getusclk, usclk_init: get/ init microsecond clock.....getusclk(3SEQ)
inittab: script for the init process.....inittab(4)
initialization init, telinit: process control.....init(1M)
files directory init.d: system level transition.....init.d(4)
list initgroups: initialize group access.....initgroups(3SEQ)
init, telinit: process control initialization.....init(1M)
reinit: run an initialization file.....reinit(1F)
bcheckrc: system initialization procedures.....bcheckrc(1M)
tput: initialize a terminal or query terminfo database.....tput(1)
s_init_barrier, s_wait_barrier: initialize barrier, wait at barrier.....s_wait_barrier(3PPS)
initgroups: initialize group access list.....initgroups(3SEQ)
/s_lock, s_clock, s_unlock: initialize, lock, unlock locks.....s_lock(3PPS)
setup: initialize system for first user.....setup(1)
inittab: script for the init process.....inittab(4)
clri: clear inode.....clri(1M)
fs, inode: format of file system volume.....fs(4)
fsirand: install random inode generation numbers.....fsirand(1M)
number of free disk blocks and inodes.....df(1M)
fscanf, sscanf: convert formatted input.....scanf(3S)
ungetc: push character back into input stream.....ungetc(3S)
fread, fwrite: binary input/output.....fread(3S)
poll: STREAMS input/output multiplexing.....poll(2)
stdio: standard buffered input/output package.....stdio(3S)
uustat: uucp status inquiry and job control.....uustat(1C)
install: install commands.....install(1M)
install: install commands.....install(1M)
fsirand: install random inode generation numbers.....fsirand(1M)
on the BOM bomverify: perform an installation consistency check based.....bomverify(1M)
on a disk installboot: installs a boot program.....installboot(1M)
product: installed products file.....product(4)
installboot: installs a boot program on a disk.....installboot(1M)
with information from/ pwconv: installs and updates /etc/shadow.....pwconv(1M)
dis386: disassemble Series 386 instructions.....dis386(3SEQ)
mailx: interactive message processing system.....mailx(1)
filesystem consistency check and interactive repair.....fsck(1M)
console: console interface.....console(7)
lp: Systech parallel line printer interface.....lp(7)

mtio: UNIX magnetic tape interface.....mtio(7)
plot: graphics interface.....plot(4)
Generic STREAMS tty driver interface.....streamtty(7)
swap: swap administrative interface.....swap(1M)
termio: general terminal interface.....termio(7)
tg: SCSI 1/2-inch magnetic tape interface.....tg(7)
tty: controlling terminal (gentty) interface.....tty(7)
wp: SSM parallel line printer interface.....wp(7)
Xylogics 472 MULITBUS magnetic tape interface.....xt(7)
timod: Transport Interface cooperating STREAMS module.....timod(7)
STREAMS module tirdwr: Transport Interface read/write interface.....tirdwr(7)
Transport Interface read/write interface STREAMS module.....tirdwr(7)
plot: graphics interface subroutines.....plot(3X)
sysadm: menu interface to do system administration.....sysadm(1)
and event tracing log: interface to STREAMS error logging.....log(7)
errorlog: internal error log file.....errorlog(4)
syskeymap: identify keys to internal menu system functions.....syskeymap(4)
asa: interpret ASA carriage control characters.....asa(1)
pathconv: search Interpreter criteria for filename.....pathconv(1F)
csh: a shell (command interpreter) with C-like syntax.....csh(1)
pipe: create an interprocess channel.....pipe(2)
ipcs: report inter-process communication facilities status.....ipcs(1)
stdipc: ftok: standard interprocess communication package.....stdipc(3C)
tfipc: interprocess communication reporter.....tfipc(1M)
ncheck: generate pathnames from i-numbers.....ncheck(1M)
streamio: STREAMS ioctl commands.....streamio(7)
ioctl: control device.....ioctl(2)
abort: generate an IOT fault.....abort(3C)
semaphore set, or shared memory ID ipcrm: remove a message queue,.....ipcrm(1)
communication facilities status ipcs: report inter-process.....ipcs(1)
/isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct,/.....ctype(3C)
isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl,/.....ctype(3C)
/iscntrl, ispunct, isprint, isgraph, isascii, tolower, toupper, toascci,/.....ctype(3C)
ttyname, isatty: find name of a terminal.....ttyname(3C)
/isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph,/.....ctype(3C)
isalpha, isalnum, isspace, iscntrl,/ isdigit, isxdigit, islower, isupper,.....ctype(3C)
/isspace, iscntrl, ispunct, isprint, isgraph, isascii, tolower, toupper,/.....ctype(3C)
file manager, updates text files in ISIS file format.....isismgr(1M)
files in ISIS file format isismgr: ISIS file manager, updates text.....isismgr(1M)
text files in ISIS file format isismgr: ISIS file manager, updates.....isismgr(1M)
isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace,/.....ctype(3C)
floating-point NaN (Not-A-Number) isnan: isnan, isnanf: test for.....isnan(3C)
isnan: isnand, isnanf: test for floating-point NaN/.....isnan(3C)
(Not-A-Number) isnan: isnand, isnanf: test for floating-point NaN.....isnan(3C)
/isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii, tolower,/.....ctype(3C)
/isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii,/.....ctype(3C)
/islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint,/.....ctype(3C)
issue identification file.....issue(4)
isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace,/.....ctype(3C)
isalnum, isspace, iscntrl,/ isdigit, isxdigit, islower, isupper, isalpha,.....ctype(3C)
getpw: get name from IUD.....getpw(3C)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions.....bessel(3M)

bessel: j0, j1, yn: Bessel functions ..... bessel(3M)
bessel: j0, j1, yn: Bessel functions ..... bessel(3M)
uustat: uucp status inquiry and job control ..... uustat(1C)
proto: prototype job file for at ..... proto(4)
join: relational database operator ..... join(1)
/erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48:/ ..... drand48(3C)
sigsetjmp, siglongjmp: non-local jumps ..... sigsetjmp(3C)
dumpconf: dump kernel auto-configuration device table ..... dumpconf(1M)
makekey: generate encryption key ..... makekey(1)
syskeymap: identify keys to internal menu system functions ..... syskeymap(4)
apropos: locate commands by keyword lookup ..... apropos(1)
system command locate: use keywords to identify an operating ..... locate(1)
killall: kill all active processes ..... killall(1M)
m_kill_procs: kill child processes ..... m_kill_procs(3PPS)
a group of processes kill: send a signal to a process or ..... kill(2)
kill: terminate a process ..... kill(1)
killall: kill all active processes ..... killall(1M)
kMBmem, kMWmem: MULITBUS memory ..... kmmem(7)
mem, kmem: core memory ..... mem(7)
kMBmem, kMWmem: MULITBUS memory ..... kmmem(7)
ksh, rksh: KornShell, a standard/restricted command and /..... ksh(1)
standard/restricted command and/ ksh, rksh: KornShell, a ..... ksh(1)
integers and long integers l3tol, ltol3: convert between 3-byte ..... l3tol(3C)
and base-64 ASCII string a64l, l64a: convert between long integer ..... a64l(3C)
filesystems labelit: provide labels for ..... labelit(1M)
labelit: provide labels for filesystems ..... labelit(1M)
awk: pattern scanning and processing language ..... awk(1)
bc: arbitrary-precision arithmetic language ..... bc(1)
efl: Extended Fortran Language ..... efl(1)
command and programming language ..... ksh(1)
pattern scanning and processing language ..... nawk(1)
command programming language ..... sh(1)
cftime: language specific strings ..... cftime(4)
chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp,/ ..... acctsh(1M)
/setspent, endspent, fgetspent, lckpwdf, ulckpwdf: get shadow/ ..... getspent(3X)
/mrand48, jrand48, srand48, seed48, lcong48: generate uniformly/ ..... drand48(3C)
files ld: link editor for object ..... see *C and Lang Tools Binder*
ld0: STREAMS tty line discipline ..... ld0(7)
ldclose, ldaclose: close a common object file ..... ldclose(3X)
a member of an archive file ldahread: read the archive header of ..... ldahread(3X)
for reading ldopen, ldaopen: open a common object file ..... ldopen(3X)
object file ldclose, ldaclose: close a common ..... ldclose(3X)
floating-point numbers frexp, ldexp, modf: manipulate parts of ..... frexp(3C)
routines ldfcn: common object file access ..... ldfcn(4)
common object file ldfhread: read the file header of a ..... ldfhread(3X)
common object file symbol table/ ldgetname: retrieve symbol name for ..... ldgetname(3X)
number entries of a common/ldlread, ldlinit, ldlitem: manipulate line ..... ldlread(3X)
entries of a/ ldlread, ldlinit, ldlitem: manipulate line number ..... ldlread(3X)
manipulate line number entries of a/ ldlread, ldlinit, ldlitem: ..... ldlread(3X)
number entries of a section of a/ ldlseek, ldnlseek: seek to line ..... ldlseek(3X)
entries of a section of a/ ldlseek, ldnlseek: seek to line number ..... ldlseek(3X)
of a section of a common/ ldrseek, ldnrseek: seek to relocation entries ..... ldrseek(3X)

section header of a/ ldshread, ldnshread: read an indexednamed.....ldshread(3X)
section of a common object/ ldsseek, ldnsseek: seek to an indexednamed.....ldsseek(3X)
header of a common object file ldohseek: seek to the optional file.....ldohseek(3X)
object file for reading ldopen, ldaopen: open a common.....ldopen(3X)
relocation entries of a section of/ ldrseek, ldnrseek: seek to.....ldrseek(3X)
indexednamed section header of a/ ldshread, ldnshread: read an.....ldshread(3X)
indexednamed section of a common/ ldsseek, ldnsseek: seek to an.....ldsseek(3X)
symbol table entry of a common/ ldtbindex: compute the index of a.....ldtbindex(3X)
table entry of a common object file ldtbread: read an indexed symbol.....ldtbread(3X)
of a common object file ldtbseek: seek to the symbol table.....ldtbseek(3X)
truncate a file to a specified length.....truncate(2SEQ)
init.d: system level transition files directory.....init.d(4)
lexical tasks lex: generate programs for simple.....lex(1)
lex: generate programs for simple lexical tasks.....lex(1)
lsearch, lfind: linear search and update.....lsearch(3C)
intro: introduction to functions and libraries.....intro(3)
find ordering relation for an object library.....see *C and Lang Tools Binder*
introduction to Parallel Programming Library.....intro(3PPS)
t_sync: synchronize transport library.....t_sync(3N)
t_alloc: allocate a library structure.....t_alloc(3N)
t_free: free a library structure.....t_free(3N)
shell (command interpreter) with C-like syntax.....csh(1)
quota: display disk usage and limits.....quota(1)
ulimit: get and set user limits.....ulimit(2)
implementation-specific constants limits: file header for.....limits(4)
returns the number of CPUs on-line.....cpus_online(3PPS)
line: read one line.....line(1)
puts its arguments on message line.....message(1F)
all but one active processor off line.....offline_all(3SEQ)
reads file and gets longest line.....readfile(1F)
establish an out-going terminal line connection.....dial(3C)
set terminal type, modes, speed, and line discipline.....getty(1M)
ld0: STREAMS tty line discipline.....ld0(7)
set terminal type, modes, speed, and line discipline.....uugetty(1M)
man: display on-line man pages.....man(1)
linenum: line number entries in a common object file.....linenum(4)
/ldlinit, ldlitem: manipulate line number entries of a common/.....ldlread(3X)
a common/ ldlseek, ldnlseek: seek to line number entries of a section of.....ldlseek(3X)
nl: line numbering filter.....nl(1)
cut: cut out selected fields of each line of a file.....cut(1)
lp: Systech parallel line printer interface.....lp(7)
wp: SSM parallel line printer interface.....wp(7)
line: read one line.....line(1)
lsearch, lfind: linear search and update.....lsearch(3C)
col: filter reverse line-feeds.....col(1)
common object file linenum: line number entries in a.....linenum(4)
readlink: read value of a symbolic link.....readlink(2SEQ)
link, unlink: link and unlink files and directories.....link(1M)
ld: link editor for object files.....see *C and Lang Tools Binder*
a.out: common assembler and link editor output.....a.out(4)
link: link to a file.....link(2)
cp, ln, mv: copy, link or move files.....cp(1)

link: link to a file ..... link(2)
symlink: make symbolic link to a file ..... symlink(2SEQ)
and directories link, unlink: link and unlink files ..... link(1M)
lint: a C program checker ..... see *C and Lang Tools Binder*
initgroups: initialize group access list ..... initgroups(3SEQ)
nlist: get entries from name list ..... nlist(3C)
setgroups: set group access list ..... setgroups(2SEQ)
varargs: handle variable argument list ..... varargs(5)
output of a varargs argument list ..... vprintf(3S)
ls: list contents of directory ..... ls(1)
ff: list filenames and statistics for a filesystem ..... ff(1M)
fsck and ncheck checklist: list of filesystems processed by ..... checklist(4)
getitems: returns a list of the currently marked menu items ..... getitems(1F)
t_listen: listen for a connect request ..... t_listen(3N)
listing list: produce C source ..... see *C and Lang Tools Binder*
xargs: construct argument list(s) and execute command ..... xargs(1)
volcopy: make literal copy of filesystem ..... volcopy(1M)
cp, ln, mv: copy, link or move files ..... cp(1)
ascftime, mktime, tzset:/ ctime, localtime, gmtime, asctime, cftime, ... ctime(3C)
apropos: locate commands by keyword lookup ..... apropos(1)
operating system command locate: use keywords to identify an ..... locate(1)
setlocale: set programs location ..... setlocale(3C)
end, etext, edata: last locations in program ..... end(3C)
plock: lock process, text, or data in memory ..... plock(2)
m_lock, m_unlock: lock, unlock locks ..... m_lock(3PPS)
s_clock, s_unlock: initialize, lock, unlock locks ..... s_lock(3PPS)
lockf: record locking on files ..... lockf(3C)
lockf: record locking on files ..... lockf(3C)
m_lock, m_unlock: lock, unlock locks ..... m_lock(3PPS)
s_unlock: initialize, lock, unlock locks ..... s_lock(3PPS)
diagnostic messages to form error log ..... dmesg(1M)
errorlog: internal error log file ..... errorlog(4)
gamma: log gamma function ..... gamma(3M)
newgrp: log in to a new group ..... newgrp(1M)
logging and event tracing log: interface to STREAMS error ..... log(7)
logarithm, power, square root/ exp, log, log10, pow, sqrt: exponential, ... exp(3M)
about RCS files rlog: print log messages and other information ..... rlog(1)
/usr/adm/loginlog: log of failed login attempts ..... loginlog(4)
sulog: log of substitute user events ..... sulog(4)
logarithm, power, square/ exp, log10, pow, sqrt: exponential, ... exp(3M)
log, log10, pow, sqrt: exponential, logarithm, power, square root/ ..... exp(3M)
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)
/usr/adm/loginlog: log of failed login attempts ..... loginlog(4)
getlogin: get login name ..... getlogin(3C)
logname: get login name ..... logname(1)
cuserid: get character login name of the user ..... cuserid(3S)
logname: return login name of user ..... logname(3X)
passwd: change login password and password attributes ..... passwd(1)
tfmenush: assign menu system as user login shell ..... tfmenush(1M)
login: sign on ..... login(1)

setting up an environment at — login time ..... profile(4)
/usr/adm/ — loginlog: log of failed login attempts ..... loginlog(4)
— logname: get login name ..... logname(1)
— logname: return login name of user ..... logname(3X)
setjmp, — longjmp: non-local goto ..... setjmp(3C)
readfile, — longline: reads file and gets longest line ..... readfile(1F)
apropos: locate commands by keyword — lookup ..... apropos(1)
make a database for fast password — lookup ..... mkpwdbm(1M)
relation for an object library — lorder: find ordering ..... see *C and Lang Tools Binder*
the LP print service — lp, cancel: send/cancel requests to ..... lp(1)
cancel: send/cancel requests to the — LP print service ..... lp(1)
lpadmin: configure the — LP print service ..... lpadmin(1M)
administer filters used with the — LP print service ..... lpfilter(1M)
administer forms used with the — LP print service ..... lpforms(1M)
information about the status of the — LP print service ..... lpstat(1)
/lpshut, lpmove: start/stop the — LP print service and move requests ..... lpsched(1M)
enable, disable: enable/disable — LP printers ..... enable(1)
accept, reject: allow or prevent — LP requests ..... accept(1M)
interface — lp: Systech parallel line printer ..... lp(7)
service — lpadmin: configure the LP print ..... lpadmin(1M)
with the LP print service — lpfilter: administer filters used ..... lpfilter(1M)
the LP print service — lpforms: administer forms used with ..... lpforms(1M)
processing module — lpmod: parallel printer canonical ..... lpmod(7)
service and move/ lpsched, lpshut, — lpmove: start/stop the LP print ..... lpsched(1M)
the LP print service and move/ — lpsched, lpshut, lpmove: start/stop ..... lpsched(1M)
print service and move/ lpsched, — lpshut, lpmove: start/stop the LP ..... lpsched(1M)
status of the LP print service — lpstat: print information about the ..... lpstat(1)
supported by the menu system — lptypes: table of printer types ..... lptypes(4)
priorities — lpusers: set printing queue ..... lpusers(1M)
srand48, seed48,/ drand48, erand48, — lrand48, nrand48, mrand48, jrand48, ... drand48(3C)
— ls: list contents of directory ..... ls(1)
update — lsearch, lfind: linear search and ..... lsearch(3C)
— lseek: move read/write file pointer ..... lseek(2)
stat, — lstat, fstat: get file status ..... stat(2)
integers and long integers l3tol, — ltol3: convert between 3-byte ..... l3tol(3C)
— m4: macro processor ..... m4(1)
u3b2, u3b5, vax: get processor type/ — machid: i386, ns32000, pdp11, u3b, ... machid(1)
show the configuration of the — machine ..... showcfg(1M)
sysi86: — machine specific functions ..... sysi86(2)
values: — machine-dependent values ..... values(5)
sgetl: access long integer data in a — machine-independent fashion ..... sputl(3X)
m4: — macro processor ..... m4(1)
mtio: UNIX — magnetic tape interface ..... mtio(7)
tg: SCSI 1/2-inch — magnetic tape interface ..... tg(7)
xt: Xylogics 472 MULITBUS — magnetic tape interface ..... xt(7)
mt: — magnetic tape manipulating program ..... mt(1)
rmail: send mail to users or read — mail ..... mail(1)
read mail — mail, rmail: send mail to users or ..... mail(1)
mail, rmail: send — mail to users or read mail ..... mail(1)
enhancement request — mailbug: submit a bug report or ..... mailbug(1)
processing system — mailx: interactive message ..... mailx(1)
groups of programs make: — maintain, update, and regenerate ..... make(1)

programs intro: introduction to ..... intro(1)
/ttyname_to_dev: convert between ..... dev_to_ttyname(3C)
regenerate groups of programs ..... make(1)
man page databases ..... makealiases(1M)
..... makedev(1M)
..... makekey(1)
databases makealiases, ... makealiases(1M)
/free, realloc, calloc, mallopt, ... malloc(3X)
memory allocator ..... malloc(3C)
mallopt, mallinfo: fast main memory/ ..... malloc(3X)
malloc, free, realloc, calloc, ... malloc(3X)
..... mallopt(3X)
..... man(1)
makealiases, makewhatis: generate ..... makealiases(1M)
man: display on-line ..... man(1)
tsearch, tfind, tdelete, twalk: ..... tsearch(3C)
hsearch, hcreate, hdestroy: ..... hsearch(3C)
endpoint t_optmgmt: ..... t_optmgmt(3N)
passmgmt: password files ..... passmgmt(1M)
sigignore, sigpause: signal ..... sigset(2)
format disks or perform other defect ..... format(1M)
file format isismgr: ISIS file ..... isismgr(1M)
mt: magnetic tape ..... mt(1)
parameters of virtual-to-physical ..... pmap(1M)
m_single, m_multi: ..... m_single(3PPS)
umask: set file-creation mode ..... umask(1)
umask: set and get file creation ..... umask(2)
regex: ..... regex(1F)
regular expression compile and ..... regexp(5)
math: ..... math(5)
..... math(5)
..... matherr(3M)
..... mbad(7)
comment section ..... mcs(1)
..... mem(7)
read the archive header of a ..... ldahread(3X)
groups: show group ..... groups(1)
memset: memory operationsmemory: ..... memory(3C)
memory operationmemory: memccpy, ... memory(3C)
operationmemory: memccpy, memchr, ... memory(3C)
memory: memccpy, memchr, memcmp, ... memory(3C)
kMBmem, kMWmem: MULITBUS ..... kmmem(7)
mem, kmem: core ..... mem(7)
lock process, text, or data in ..... plock(2)
create security table in shared ..... tasktbl(1M)
malloc, free, realloc, calloc: main ..... malloc(3C)
calloc, mallopt, mallinfo: fast main ..... malloc(3X)
shmalloc, shrealloc, shfree: shared ..... shmalloc(3PPS)
valloc: aligned ..... valloc(3SEQ)
shmctl: shared ..... shmctl(2)
consmem, sm, ss: console ..... consmem(7)
dump: standalone ..... dump(8)
spawn new process in a virtual ..... vfork(2)

queue, semaphore set, or shared memory ID ..... ipcrm(1)
memcpy, memset: memory operations memory: memccpy, memchr, memcmp, ... memory(3C)
memchr, memcmp, memcpy, memset: memory operations ..... memory(3C)
shmop, shmat, shmdt: shared memory operations ..... shmop(2)
vmtune: tune system-wide virtual- memory parameters ..... vmtune(1M)
shmget: get shared memory segment identifier ..... shmget(2)
vm_ctl: examine and change virtual memory tuning parameters ..... vm_ctl(2SEQ)
memccpy, memchr, memcmp, memcpy, memset: memory operations ..... memory(3C)
addcmdopt: add a command option to a menu ..... addcmdopt(1M)
add a form/SELECT window option to a menu ..... addformopt(1M)
addmenuopt: add a menu option to a menu ..... addmenuopt(1M)
sysadm: menu interface to do system administration ..... sysadm(1)
a list of the currently marked menu items ..... getitems(1F)
menu: menu processor ..... menu(1M)
menumsg: route user message to menu message queue ..... menumsg(1M)
rmvopt: remove a menu option ..... rmvopt(1M)
addmenuopt: add a menu option to a menu ..... addmenuopt(1M)
menu: menu processor ..... menu(1M)
of printer types supported by the menu system ..... lptypes(4)
tfmenush: assign menu system as user login shell ..... tfmenush(1M)
syskeymap: identify keys to internal menu system functions ..... syskeymap(4)
message queue menumsg: route user message to menu ..... menumsg(1M)
for generating/modifying ASSIST menus or command forms ..... astgen(1)
merge: three-way file merge ..... merge(1)
sort: sort and/or merge files ..... sort(1)
acctmerg: merge or add total accounting files ..... acctmerg(1M)
rcsmerge: merge RCS revisions ..... rcsmerge(1)
subsequent lines of one file paste: merge same lines of several files or ..... paste(1)
merge: three-way file merge ..... merge(1)
mesg: permit or deny messages ..... mesg(1)
t_error: produce error message ..... t_error(3N)
tfnotify: send completion message ..... tfnotify(1M)
msgctl: message control operations ..... msgctl(2)
message: puts its arguments on message line ..... message(1F)
getmsg: get next message off a stream ..... getmsg(2)
putmsg: send a message on a stream ..... putmsg(2)
msgop: message operations ..... msgop(2)
mailx: interactive message processing system ..... mailx(1)
message line message: puts its arguments on ..... message(1F)
menumsg: route user message to menu message queue ..... menumsg(1M)
msgget: get message queue ..... msgget(2)
shared memory ID ipcrm: remove a message queue, semaphore set, or ..... ipcrm(1)
errormsg: error message text file ..... errormsg(4)
menumsg: route user message to menu message queue ..... menumsg(1M)
motd: message-of-the-day file ..... motd(4)
mesg: permit or deny messages ..... mesg(1)
sys_errlist, sys_nerr: system error messages ..... perror(3C)
strace: print STREAMS trace messages ..... strace(1M)
rlog: print log messages and other information about RCS files ..... rlog(1)
dmesg: collect system diagnostic messages to form error log ..... dmesg(1M)
parallel m_fork: execute a subprogram in ..... m_fork(3PPS)
identification m_get_myid: return process ..... m_get_myid(3PPS)

number of child processes m_get_numprocs: return ..... m_get_numprocs(3PPS)
getusclk, usclk_init: get/init microsecond clock ..... getusclk(3SEQ)
usclk: 32-bit microsecond counter ..... usclk(7)
aging, PFF, swapping, or processor migration disabled ..... noage(1M)
clone: open any minor device on a STREAMS driver ..... clone(7)
/convert between major/ minor numbers and tty names ..... dev_to_ttyname(3C)
/acctwtmp: overview of accounting and miscellaneous accounting commands ..... acct(1M)
intro: introduction to miscellany ..... intro(5)
mkdir: make a directory ..... mkdir(2)
mkdir: make directories ..... mkdir(1)
mkfifo: make a FIFO special file ..... mkfifo(3C)
mkfs: construct a filesystem ..... mkfs(1M)
m_kill_procs: kill child processes ..... m_kill_procs(3PPS)
mknod: build special file ..... mknod(1M)
special or ordinary file mknod: make a directory, or a ..... mknod(2)
password lookup mkpwdbm: make a database for fast ..... mkpwdbm(1M)
mktemp: make a unique filename ..... mktemp(3C)
/gmtime, asctime, cftime, ascftime, mktime, tzset: convert date and time/ ..... ctime(3C)
mkvtoc: populate VTOC on disks ..... mkvtoc(1M)
m_lock, m_unlock: lock, unlock locks ..... m_lock(3PPS)
processs address space mmap: map an open file into the ..... mmap(2SEQ)
section m_single, m_multi: mark single-process code ..... m_single(3PPS)
m_next: increment global counter ..... m_next(3PPS)
mnttab: mounted filesystem table ..... mnttab(4)
configuration file mod_config: format of a module ..... mod_config(4)
chmod: change mode ..... chmod(1)
umask: set file-creation mode mask ..... umask(1)
chmod: change mode of file ..... chmod(2)
cpset: copy file and set mode, user, and group IDs ..... cpset(1)
getty: set terminal type, modes, speed, and line discipline ..... getty(1M)
uugetty: set terminal type, modes, speed, and line discipline ..... uugetty(1M)
frexp, ldexp, modf: manipulate parts of floating-point/ ..... frexp(3C)
utime: set file access and modification times ..... utime(2)
touch: update access and modification times of a file ..... touch(1)
astgen: program for generating/ modifying ASSIST menus or command/ ..... astgen(1)
printer canonical processing module ..... lpmod(7)
SSM: System Services Module ..... ssm(7)
Interface cooperating STREAMS module ..... timod(7)
read/write interface STREAMS module ..... tirdwr(7)
mod_config: format of a module configuration file ..... mod_config(4)
/ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily/ ..... acctsh(1M)
helplog: monitor help actions ..... helplog(4)
monitor: monitor system activity ..... monitor(1M)
monitor: prepare execution profile ..... monitor(3C)
monitor: monitor system activity ..... monitor(1M)
a time more: view a file one full screen at ..... more(1)
motd: message-of-the-day file ..... motd(4)
mount: mount a filesystem ..... mount(2)
mount, umount: mount and unmount filesystems ..... mount(1M)
mount: mount a filesystem ..... mount(2)
setmnt: establish mount table ..... setmnt(1M)
filesystems mount, umount: mount and unmount ..... mount(1M)

mountall, umountall: mount, unmount multiple filesystems.....mountall(1M)
multiple filesystems mountall, umountall: mount, unmount.....mountall(1M)
mnttab: mounted filesystem table.....mnttab(4)
mvdir: move a directory.....mvdir(1M)
cp, ln, mv: copy, link or move files.....cp(1)
lseek: move read/write file pointer.....lseek(2)
start/stop the LP print service and move requests.....lpsched(1M)
and resume child process execution m_park_procs, m_rele_procs: susp/....m_park_procs(3PPS)
ptrace, mptrace: process trace facility.....ptrace(2)
drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48,/.....drand48(3C)
child process/ m_park_procs, m_rele_procs: suspend and resume....m_park_procs(3PPS)
processes m_set_procs: set number of child.....m_set_procs(3PPS)
msgctl: message control operations.....msgctl(2)
msgget: get message queue.....msgget(2)
msgop: message operations.....msgop(2)
single-process code section m_single, m_multi: mark.....m_single(3PPS)
m_sync: check in at barrier.....m_sync(3PPS)
program mt: magnetic tape manipulating.....mt(1)
st: Systech MTI-800/1600/1650 terminal multiplexor driver.....st(7)
mtio: UNIX magnetic tape interface.....mtio(7)
mbad: MULITBUS adapter.....mbad(7)
xt: Xylogics 472 MULITBUS magnetic tape interface.....xt(7)
kMBmem, kMWmem: MULITBUS memory.....kmmem(7)
mountall, umountall: mount, unmount multiple filesystems.....mountall(1M)
poll: STREAMS input/output multiplexing.....poll(2)
Systech MTI-800/1600/1650 terminal multiplexor driver.....st(7)
xs: Xylogics 781 terminal multiplexor driver.....xs(7)
tmp_ctl: multi-processor control.....tmp_ctl(2SEQ)
rc2: run commands performed for multi-user environment.....rc2(1M)
m_lock, m_unlock: lock, unlock locks.....m_lock(3PPS)
address space munmap: unmap part of the processs.....munmap(2SEQ)
cp, ln, mv: copy, link or move files.....cp(1)
mvdir: move a directory.....mvdir(1M)
devnm: device name.....devnm(1M)
getenv: return value for environment name.....getenv(3C)
getgeombyname: get disk geometry by name.....getgeombyname(3SEQ)
getlogin: get login name.....getlogin(3C)
logname: get login name.....logname(1)
pwd: working directory name.....pwd(1)
tmpnam, tempnam: create a name for a temporary file.....tmpnam(3S)
table/ ldgetname: retrieve symbol name for common object file symbol.....ldgetname(3X)
getpw: get name from IUD.....getpw(3C)
nlist: get entries from name list.....nlist(3C)
rename: change the name of a file.....rename(2)
ttyname, isatty: find name of a terminal.....ttyname(3C)
uname: get name of current operating system.....uname(2)
uname: print name of current system.....uname(1)
tty: get the name of the terminal.....tty(1)
cuserid: get character login name of the user.....cuserid(3S)
logname: return login name of user.....logname(3X)
/ldnshread: read an indexed named section header of a common object/.....ldshread(3X)
/ldnsseek: seek to an indexed named section of a common object file.....ldsseek(3X)

between major/minor numbers and tty names.....dev_to_ttyname(3C)
id: print user and group IDs and names.....id(1M)
term: conventional names for terminals.....term(5)
isnanf: test for floating-point NaN (Not-A-Number).....isnan(3C)
processing language nawk: pattern scanning and .....nawk(1)
and indirect blocks nblocks: calculate number of blocks.....nblocks(3SEQ)
of filesystems processed by fsck and ncheck.....checklist(4)
i-numbers ncheck: generate pathnames from .....ncheck(1M)
uucp: networking support files.....uucp(4)
file newform: change the format of a text .....newform(1)
newfs: construct a new file system .....newfs(1M)
newgrp: log in to a new group.....newgrp(1M)
news: print news items.....news(1)
news: print news items.....news(1)
getmsg: get next message off a stream.....getmsg(2)
nice: change priority of a process.....nice(2)
nice: run a command at low priority .....nice(1)
nl: line numbering filter.....nl(1)
nlist: get entries from name list .....nlist(3C)
object file nm: print name list of.....see *C and Lang Tools Binder*
command with priority aging, PFF/ noage, nopff, noswap, onproc: run a .....noage(1M)
hangups and quits nohup: run a command immune to .....nohup(1)
setjmp, longjmp: non-local goto.....setjmp(3C)
sigsetjmp, siglongjmp: non-local jumps.....sigsetjmp(3C)
with priority aging, PFF/ noage, nopff, noswap, onproc: run a command.....noage(1M)
priority aging, PFF/ noage, nopff, noswap, onproc: run a command with .....noage(1M)
test for floating-point NaN (Not-A-Number).....isnan(3C)
seed48/ drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, ...drand48(3C)
deroff: remove nroff, tbl, and eqn constructs.....deroff(1)
vax: get processor/ machid: i386, ns32000, pdp11, u3b, u3b2, u3b5,.....machid(1)
null: the null file.....null(7)
null: the null file .....null(7)
/ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct,/ .....acctsh(1M)
obtain the prime factors of a number.....factor(1)
getfrm: returns the current frame number.....getfrm(1F)
test for floating-point NaN (Not-A- Number) .....isnan(3C)
convert string to double-precision number.....strtod(3C)
linenum: line number entries in a common object file.....linenum(4)
/ldlinit, ldlitem: manipulate line number entries of a common object/.....ldlread(3X)
ldlseek, ldnlseek: seek to line number entries of a section of a/ .....ldlseek(3X)
rand, srand: simple random- number generator.....rand(3C)
nblocks: calculate number of blocks and indirect blocks.....nblocks(3SEQ)
m_get_numprocs: return number of child processes .....m_get_numprocs(3PPS)
m_set_procs: set number of child processes.....m_set_procs(3PPS)
cpus_online: returns the number of CPUs on-line .....cpus_online(3PPS)
df: report number of free disk blocks and inodes.....df(1M)
fcvt, gcvt: convert floating-point number to string .....ecvt(3C)
nl: line numbering filter.....nl(1)
uniformly distributed pseudorandom numbers.....drand48(3C)
manipulate parts of floating-point numbers .....frexp(3C)
install random inode generation numbers.....fsirand(1M)
to system calls and error numbers .....intro(2)

ncheck: generate pathnames from i- numbers.....ncheck(1M)
/convert between major/minor numbers and tty names.....dev_to_ttyname(3C)
cprs: compress an object file.....see *C and Lang Tools Binder*
strstr: find first occurrence of character.....strstr(3C)
od: octal dump.....od(1)
od: octal dump.....od(1)
offline: bring processors online and offline.....online(1M)
online, offline: bring processors online and offline.....online(1M)
processor off line offline_all: take all but one active.....offline_all(3SEQ)
online, offline: bring processors online and offline.....online(1M)
online and offline online, offline: bring processors.....online(1M)
aging, PFF,/ noage, nopff, noswap, onproc: run a command with priority.....noage(1M)
ldopen, ldaopen: open a common object file for reading.....ldopen(3X)
fopen, freopen, fdopen: open a stream.....fopen(3S)
clone: open any minor device on a STREAMS driver.....clone(7)
dup: duplicate an open file descriptor.....dup(2)
dup2: duplicate an open file descriptor.....dup2(2)
mmap: map an open file into the processs address space.....mmap(2SEQ)
open: open for reading or writing.....open(2)
open: open for reading or writing.....open(2)
rewinddir, closedir/ directory: opendir, readdir, telldir, seekdir,.....directory(3X)
a standalone program or bring up the operating system.....boot(8)
run commands performed to stop the operating system.....rc0(1M)
savecore: save a core dump of the operating system.....savecore(1M)
uname: get name of current operating system.....uname(2)
locate: use keywords to identify an operating system command.....locate(1)
assist: assistance using operating system commands.....assist(1)
starter: information about the operating system for beginning users.....starter(1)
help: operating system Help Facility.....help(1)
glossary: definitions of common operating system terms and symbols.....glossary(1)
join: relational database operator.....join(1)
curses: terminal screen handling and optimization package.....curses(3X)
rmvopt: remove a menu option.....rmvopt(1M)
getopt: get option letter from argument vector.....getopt(3C)
addcmdopt: add a command option to a menu.....addcmdopt(1M)
addformopt: add a form/SELECT window option to a menu.....addformopt(1M)
addmenuopt: add a menu option to a menu.....addmenuopt(1M)
object file ldohseek: seek to the optional file header of a common.....ldohseek(3X)
fcntl: file control options.....fcntl(5)
getopt: parse command options.....getopt(1)
getopts, getoptcvt: parse command options.....getopts(1)
stty: set the options for a terminal.....stty(1)
t_optmgmt: manage options for a transport endpoint.....t_optmgmt(3N)
make a directory, or a special or ordinary file.....mknod(2)
architecture information ott: files that hold object.....ott(4)
common assembler and link editor output.....a.out(4)
echo: put string on virtual output.....echo(1F)
fread, fwrite: binary input/ output.....fread(3S)
fprintf, sprintf: print formatted output.....printf(3S)
poll: STREAMS input/ output multiplexing.....poll(2)
/vfprintf, vsprintf: print formatted output of a varargs argument list.....vprintf(3S)
stdio: standard buffered input/ output package.....stdio(3S)

setpgrp: run command in its own process group.....setpgrp(1)
chown: change owner and group of a file.....chown(2)
chown, chgrp: change owner or group.....chown(1)
expand files pack, pcat, unpack: compress and.....pack(1)
makewhatis: generate man page databases.....makealiases(1M)
getpagesize: get system page size.....getpagesize(2SEQ)
4014: paginator for the Tektronix 4014 terminal.....4014(1)
m_fork: execute a subprogram in parallel.....m_fork(3PPS)
lp: Systech parallel line printer interface.....lp(7)
wp: SSM parallel line printer interface.....wp(7)
lpmod: parallel printer canonical processing module.....lpmod(7)
intro: introduction to Parallel Programming Library.....intro(3PPS)
and change virtual memory tuning parameters.....vm_ctl(2SEQ)
tune system-wide virtual-memory parameters.....vmtune(1M)
pmap: display or alter parameters of virtual-to-physical mapping/.....pmap(1M)
get process, process group, and parent process IDs.....getpid(2)
getopt: parse command options.....getopt(1)
getopts, getoptcvt: parse command options.....getopts(1)
cmpt: compatibility partitioning driver.....cmpt(7)
vtoc: VTOC partitioning driver.....vtoc(7)
frexp, ldexp, modf: manipulate parts of floating-point numbers.....frexp(3C)
passmgmt: password files management.....passmgmt(1M)
with information from /etc/ passwd.....pwconv(1M)
password attributes passwd: change login password and.....passwd(1)
passwd: password file.....passwd(4)
getpass: read a password.....getpass(3C)
crypt: password and file encryption functions.....crypt(3X)
passwd: change login password and password attributes.....passwd(1)
passwd: change login password and password attributes.....passwd(1)
pwdhd: daemon to oversee password database creation.....pwdhd(1M)
pwfix: remove incorrectly formatted password entries.....pwfix(1M)
passwd: password file.....passwd(4)
shadow: restricted password file.....shadow(4)
setpwent, endpwent, fgetpwent: get password file entry.....getpwent(3C)
lckpwdf, ulckpwdf: get shadow password file entry.....getspent(3X)
putpwent: write password file entry.....putpwent(3C)
putspent: write shadow password file entry.....putspent(3X)
converts from a two to a one password file scheme.....pwunconv(1M)
passmgmt: password files management.....passmgmt(1M)
mkpwdbm: make a database for fast password lookup.....mkpwdbm(1M)
pwck, grpck: password/group file checkers.....pwck(1M)
files or subsequent lines of one/ paste: merge same lines of several.....paste(1)
bp: binary patcher.....bp(1M)
pathalias: alias file for FACE.....alias(4)
configurable pathname variables pathconf, fpathconf: get.....pathconf(2)
criteria for filename pathconv: search Interpreter.....pathconv(1F)
getcwd: get pathname of current working directory.....getcwd(3C)
fpathconf: get configurable pathname variables.....pathconf(2)
dirname: deliver portions of pathnames.....basename(1)
ncheck: generate pathnames from i-numbers.....ncheck(1M)
sweepon, sweepoff: enable/disable patrol seeks on DCC disks.....sweepon(1M)
grep: search a file for a pattern.....grep(1)

awk: pattern scanning and processing language ..... awk(1)
nawk: pattern scanning and processing language ..... nawk(1)
egrep: search a file for a pattern using full regular/ ..... egrep(1)
regex: match patterns against a string ..... regex(1F)
pause: suspend process until signal ..... pause(2)
pack, pcat, unpack: compress and expand files ..... pack(1)
popen, pclose: initiate pipe to/from a process ..... popen(3S)
pdbx, dbx: parallel/ ..... see *C and Lang Tools Binder*
machid: i386, ns32000, pdp11, u3b, u3b2, u3b5, vax: get processor/ ..... machid(1)
sigpending: examine pending signals ..... sigpending(2)
check the uucp directories and permissions file ..... uucheck(1M)
acct: per-process accounting file format ..... acct(4)
acctcms: command summary from per-process accounting records ..... acctcms(1M)
sys_nerr: system error messages perror, errno, sys_errlist, ... perror(3C)
run a command with priority aging, PFF, swapping, or processor/ ..... noage(1M)
pg: file perusal filter for CRTs ..... pg(1)
virtual address space pmap: map physical addresses into processs ..... pmap(7)
or alter parameters of virtual-to- physical mapping devices ..... pmap(1M)
pipe: create an interprocess channel ..... pipe(2)
sp: STREAMS pipe device ..... sp(7)
tee: pipe fitting ..... tee(1)
popen, pclose: initiate pipe to/from a process ..... popen(3S)
in memory plock: lock process, text, or data ..... plock(2)
plot: graphics interface ..... plot(4)
plot: graphics interface subroutines ..... plot(3X)
virtual-to-physical mapping devices pmap: display or alter parameters of ..... pmap(1M)
processs virtual address space pmap: map physical addresses into ..... pmap(7)
/fpsetsticky: IEEE floating- point environment control ..... fpgetround(3C)
isnand, isnanf: test for floating- point NaN (Not-A-Number) ..... isnan(3C)
ecvt, fcvt, gcvt: convert floating- point number to string ..... ecvt(3C)
modf: manipulate parts of floating- point numbers ..... frexp(3C)
lseek: move read/write file pointer ..... lseek(2)
rewind, ftell: reposition a file pointer in a stream ..... fseek(3S)
multiplexing poll: STREAMS input/output ..... poll(2)
a process popen, pclose: initiate pipe to/from ..... popen(3S)
banner: make posters ..... banner(1)
power, square root/ exp, log, log10, pow, sqrt: exponential, logarithm, ... exp(3M)
pow, sqrt: exponential, logarithm, power, square root functions ..... exp(3M)
pr: print files ..... pr(1)
/dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, shutacct,/ ..... acctsh(1M)
/lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, shutacct, startup,/ ..... acctsh(1M)
bc: arbitrary- precision arithmetic language ..... bc(1)
atof: convert string to double- precision number ..... strtod(3C)
.environ, .pref, .variables: FACE files ..... env(4)
monitor: prepare execution profile ..... monitor(3C)
cpp: the C language preprocessor ..... see *C and Lang Tools Binder*
factor: obtain the prime factors of a number ..... factor(1)
gps: graphical primitive string, format of graphical files ..... gps(4)
types: primitive system data types ..... types(5)
prs: print an SCCS file ..... prs(1)
date: print and set the date ..... date(1)
cal: print calendar ..... cal(1)

sum: print checksum and block count of a file.....sum(1)
sact: print current SCCS file editing activity.....sact(1)
cat: concatenate and print files.....cat(1)
pr: print files.....pr(1)
printf, fprintf, sprintf: print formatted output.....printf(3S)
vprintf, vfprintf, vsprintf: print formatted output of a varargs/.....vprintf(3S)
of the LP print service lpstat: print information about the status.....lpstat(1)
information about RCS files rlog: print log messages and other.....rlog(1)
object file nm: print name list of.....see *C and Lang Tools Binder*
uname: print name of current system.....uname(1)
news: print news items.....news(1)
infocmp: compare or print out terminfo descriptions.....infocmp(1M)
acctcom: search and print process accounting files.....acctcom(1)
of object files size: print section sizes.....see *C and Lang Tools Binder*
send/cancel requests to the LP print service.....lp(1)
lpadmin: configure the LP print service.....lpadmin(1M)
administer filters used with the LP print service.....lpfilter(1M)
administer forms used with the LP print service.....lpforms(1M)
about the status of the LP print service.....lpstat(1)
/lpshut, lpmove: start/stop the LP print service and move requests.....lpsched(1M)
strace: print STREAMS trace messages.....strace(1M)
prtvtoc: print the VTOC of a block device.....prtvtoc(1M)
prtvtoc: print the VTOC of a standalone disk device.....prtvtoc(8)
id: print user and group IDs and names.....id(1M)
lpmod: parallel printer canonical processing module.....lpmod(7)
lp: Systech parallel line printer interface.....lp(7)
wp: SSM parallel line printer interface.....wp(7)
lptypes: table of printer types supported by the menu system.....lptypes(4)
enable, disable: enable/disable LP printers.....enable(1)
formatted output printf, fprintf, sprintf: print.....printf(3S)
lpusers: set printing queue priorities.....lpusers(1M)
lpusers: set printing queue priorities.....lpusers(1M)
nice: run a command at low priority.....nice(1)
/noswap, onproc: run a command with priority aging, PFF, swapping, or/.....noage(1M)
nice: change priority of a process.....nice(2)
brk, sbrk: change private data segment size.....brk(3PPS)
process attributes proc_ctl: control over various.....proc_ctl(2SEQ)
codestroy: communicate to a process.....coproc(1F)
exit, _exit: terminate process.....exit(2)
fork: create a new process.....fork(2)
inittab: script for the init process.....inittab(4)
kill: terminate a process.....kill(1)
nice: change priority of a process.....nice(2)
pclose: initiate pipe to/from a process.....popen(3S)
wait: await completion of process.....wait(1)
acct: enable or disable process accounting.....acct(2)
acctprc1, acctprc2: process accounting.....acctprc(1M)
acct: per- process accounting file format.....acct(4)
acctcom: search and print process accounting files.....acctcom(1)
acctcms: command summary from per- process accounting records.....acctcms(1M)
alarm: set a process alarm clock.....alarm(2)
times: get process and child process times.....times(2)

proc_ctl: control over various process attributes.....proc_ctl(2SEQ)
m_single, m_multi: mark single- process code section.....m_single(3PPS)
ipcs: report inter- process communication facilities status.....ipcs(1)
init, telinit: process control initialization.....init(1M)
timex: time a command; report process data and system activity.....timex(1)
suspend and resume child process execution.....m_park_procs(3PPS)
setpgrp: run command in its own process group.....setpgrp(1)
/getpgrp, getppid: get process, process group, and parent process IDs.....getpid(2)
setpgrp: set process group ID.....setpgrp(2)
setsid: create session and set process group ID.....setsid(2)
tcgetpgrp: get foreground process group ID.....tcgetpgrp(2)
tcsetpgrp: set foreground process group ID.....tcsetpgrp(2)
m_get_myid: return process identification.....m_get_myid(3PPS)
process, process group, and parent process IDs.....getpid(2)
vfork: spawn new process in a virtual memory efficient way.....vfork(2)
kill: send a signal to a process or a group of processes.....kill(2)
getpid, getpgrp, getppid: get process, process group, and parent/.....getpid(2)
ps: report process status.....ps(1)
plock: lock process, text, or data in memory.....plock(2)
times: get process and child process times.....times(2)
tmp_affinity: bind a process to a processor.....tmp_affinity(2SEQ)
wait, waitpid: wait for a child process to stop or terminate.....wait(2)
ptrace, mptrace: process trace facility.....ptrace(2)
pause: suspend process until signal.....pause(2)
vsig: synchronize a co- process with the controlling FMLI object.....vsig(1F)
checklist: list of filesystems processed by fsck and ncheck.....checklist(4)
a signal to a process or a group of processes.....kill(2)
killall: kill all active processes.....killall(1M)
return number of child processes.....m_get_numprocs(3PPS)
m_kill_procs: kill child processes.....m_kill_procs(3PPS)
m_set_procs: set number of child processes.....m_set_procs(3PPS)
fuser: identify processes using a file or file structure.....fuser(1M)
m4: macro processor.....m4(1)
menu: menu processor.....menu(1M)
tmp_affinity: bind a process to a processor.....tmp_affinity(2SEQ)
tmp_ctl: multi- processor control.....tmp_ctl(2SEQ)
priority aging, PFF, swapping, or processor migration disabled.....noage(1M)
offline_all: take all but one active processor off line.....offline_all(3SEQ)
pdp11, u3b, u3b2, u3b5, vax: get processor type truth value.....machid(1)
online, offline: bring processors online and offline.....online(1M)
mmap: map an open file into the processs address space.....mmap(2SEQ)
munmap: unmap part of the processs address space.....munmap(2SEQ)
pmap: map physical addresses into processs virtual address space.....pmap(7)
product: installed products file.....product(4)
prof: display profile data.....prof(1)
prof: profile within a function.....prof(5)
profil: execution time profile.....profil(2)
monitor: prepare execution profile.....monitor(3C)
profil: execution time profile.....profil(2)
prof: display profile data.....prof(1)
at login time profile: setting up an environment.....profile(4)
prof: profile within a function.....prof(5)

end, etext, edata: last locations in program ..... end(3C)
mt: magnetic tape manipulating program ..... mt(1)
prompt: prompt program ..... prompt(1M)
sdiff: side-by-side difference program ..... sdiff(1)
STREAMS error logger cleanup program ..... strclean(1M)
units: conversion program ..... units(1)
for the uucp file transport program ..... uusched(1M)
assert: verify program assertion ..... assert(3X)
ASSIST menus or command/ astgen: program for generating/modifying ..... astgen(1)
uucico: file transport program for the uucp system ..... uucico(1M)
installboot: installs a boot program on a disk ..... installboot(1M)
boot: run a standalone program or bring up the operating system ..... boot(8)
a standard/restricted command/ programming language ..... ksh(1)
the standard/restricted command programming language ..... sh(1)
intro: introduction to Parallel Programming Library ..... intro(3PPS)
maintenance commands and application programs ..... intro(1)
update, and regenerate groups of programs ..... make(1)
lex: generate programs for simple lexical tasks ..... lex(1)
setlocale: set programs location ..... setlocale(3C)
prompt: prompt program ..... prompt(1M)
prompt: prompt program ..... prompt(1M)
proto: prototype job file for at ..... proto(4)
t_getinfo: get protocol-specific service information ..... t_getinfo(3N)
proto: prototype job file for at ..... proto(4)
prs: print an SCCS file ..... prs(1)
/monacct, nulladm, prctmp, prdaily, prtacct, shutacct, startup,/ ..... acctsh(1M)
device prtvtoc: print the VTOC of a block ..... prtvtoc(1M)
standalone disk device prtvtoc: print the VTOC of a ..... prtvtoc(8)
ps: report process status ..... ps(1)
streampty: STREAMS pseudo terminal driver ..... streampty(7)
getpseudotty: get pseudo terminals ..... getpseudotty(3SEQ)
cn: pseudoconsole driver ..... cn(7)
generate uniformly distributed pseudorandom numbers ..... drand48(3C)
time, ptime: time a command ..... time(1)
facility ptrace, mptrace: process trace ..... ptrace(2)
uuto, uupick: public UNIX-to-UNIX system file copy ..... uuto(1C)
ungetc: push character back into input stream ..... ungetc(3S)
puts, fputs: put a string on a stream ..... puts(3S)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
getdents: read directory entries and put in a file system independent/ ..... getdents(2)
echo: put string on virtual output ..... echo(1F)
character or word on a stream putc, putchar, fputc, putw: put ..... putc(3S)
or word on a stream putc, putchar, fputc, putw: put character ..... putc(3S)
environment putenv: change or add value to ..... putenv(3C)
putmsg: send a message on a stream ..... putmsg(2)
putpwent: write password file entry ..... putpwent(3C)
stream puts, fputs: put a string on a ..... puts(3S)
message: puts its arguments on message line ..... message(1F)
entry putspent: write shadow password file ..... putspent(3X)
getut: getutent, getutid, getutline, pututline, setutent, endutent,/ ..... getut(3C)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
checkers pwck, grpck: password/group file ..... pwck(1M)

/etc/shadow with information from/ pwconv: installs and updates ..... pwconv(1M)
pwd: working directory name ..... pwd(1)
database creation pwdhd: daemon to oversee password ..... pwdhd(1M)
password entries pwfix: remove incorrectly formatted ..... pwfix(1M)
one password file scheme pwunconv: converts from a two to a ..... pwunconv(1M)
qsort: quicker sort ..... qsort(3C)
tput: initialize a terminal or query terminfo database ..... tput(1)
route user message to menu message queue ..... menumsg(1M)
msgget: get message queue ..... msgget(2)
lpusers: set printing queue priorities ..... lpusers(1M)
ipcrm: remove a message queue, semaphore set, or shared memory ID ..... ipcrm(1)
quotacheck: filesystem quota consistency checker ..... quotacheck(1M)
quota: display disk usage and limits ..... quota(1)
consistency checker quotacheck: filesystem quota ..... quotacheck(1M)
quotactl: manipulate disk quotas ..... quotactl(2SEQ)
quotaon, quotaoff: turn filesystem quotas on and off ..... quotaon(1M)
quotas on and off quotaon, quotaoff: turn filesystem ..... quotaon(1M)
edquota: edit user quotas ..... edquota(1M)
quotactl: manipulate disk quotas ..... quotactl(2SEQ)
repquota: summarize quotas for a filesystem ..... repquota(1M)
quotaon, quotaoff: turn filesystem quotas on and off ..... quotaon(1M)
generator rand, srand: simple random-number ..... rand(3C)
fsirand: install random inode generation numbers ..... fsirand(1M)
rand, srand: simple random-number generator ..... rand(3C)
fsplit: split f77, ratfor, or efl files ..... fsplit(1)
ratfor: rational Fortran dialect ..... ratfor(1)
ratfor: rational Fortran dialect ..... ratfor(1)
scripts rc: reboot and shutdown command ..... rc(4)
the operating system rc0: run commands performed to stop ..... rc0(1M)
multi-user environment rc2: run commands performed for ..... rc2(1M)
rcs: change RCS file attributes ..... rcs(1)
rcsintro: introduction to RCS commands ..... rcsintro(1)
rcsfile: format of RCS file ..... rcsfile(4)
rcs: change RCS file attributes ..... rcs(1)
sccstorcs: build RCS file from SCCS file ..... sccstorcs(1)
messages and other information about RCS files ..... rlog(1)
ci: check in RCS revisions ..... ci(1)
co: check out RCS revisions ..... co(1)
rcsdiff: compare RCS revisions ..... rcsdiff(1)
rcsmerge: merge RCS revisions ..... rcsmerge(1)
rcsdiff: compare RCS revisions ..... rcsdiff(1)
rcsfile: format of RCS file ..... rcsfile(4)
commands rcsintro: introduction to RCS ..... rcsintro(1)
rcsmerge: merge RCS revisions ..... rcsmerge(1)
getpass: read a password ..... getpass(3C)
of a common object file ldtbread: read an indexed symbol table entry ..... ldtbread(3X)
header of a/ ldshread, ldnshread: read an indexednamed section ..... ldshread(3X)
structure read_constab: read constab entries into a ..... read_constab(3SEQ)
file system independent/ getdents: read directory entries and put in a ..... getdents(2)
read: read from file ..... read(2)
mail, rmail: send mail to users or read mail ..... mail(1)
line: read one line ..... line(1)

	read:	read from file ..... read(2)
of an archive file	ldahread:	read the archive header of a member ..... ldahread(3X)
object file	ldfhread:	read the file header of a common ..... ldfhread(3X)
	readlink:	read value of a symbolic link ..... readlink(2SEQ)
	readv:	read vector from file ..... readv(2SEQ)
into a structure	read_constab:	read constab entries ..... read_constab(3SEQ)
directory: opendir,		readdir, telldir, seekdir, rewinddir,/ ..... directory(3X)
gets longest line		readfile, longline: reads file and ..... readfile(1F)
open a common object file for		reading ..... ldopen(3X)
open: open for		reading or writing ..... open(2)
link		readlink: read value of a symbolic ..... readlink(2SEQ)
readfile, longline:		reads file and gets longest line ..... readfile(1F)
		readv: read vector from file ..... readv(2SEQ)
lseek: move		read/write file pointer ..... lseek(2)
tirdwr: Transport Interface		read/write interface STREAMS module ..... tirdwr(7)
/get real user, effective user,		real group, and effective group IDs ..... getuid(2)
/geteuid, getgid, getegid: get		real user, effective user, real/ ..... getuid(2)
malloc, free,		realloc, calloc: main memory allocator ..... malloc(3C)
fast main memory/	malloc, free,	realloc, calloc, mallopt, mallinfo: ..... malloc(3X)
rc:		reboot and shutdown command scripts ..... rc(4)
t_rcvudata:		receive a data unit ..... t_rcvudata(3N)
t_rcvuderr:		receive a unit data error indication ..... t_rcvuderr(3N)
over a connection	t_rcv:	receive data or expedited data sent ..... t_rcv(3N)
connect request	t_rcvconnect:	receive the confirmation from a ..... t_rcvconnect(3N)
lockf:		record locking on files ..... lockf(3C)
ed,		red: text editor ..... ed(1)
setcolor:		redefine or create a color ..... setcolor(1F)
dltsectbl:		refresh security table ..... dltsectbl(1M)
regular expression		regcmp, regex: compile and execute ..... regcmp(3X)
		regcmp: regular expression compile ..... regcmp(1)
make: maintain, update, and		regenerate groups of programs ..... make(1)
expression	regcmp,	regex: compile and execute regular ..... regcmp(3X)
string		regex: match patterns against a ..... regex(1F)
and match routines		regexp: regular expression compile ..... regexp(5)
regcmp, regex: compile and execute		regular expression ..... regcmp(3X)
regcmp:		regular expression compile ..... regcmp(1)
routines	regexp:	regular expression compile and match ..... regexp(5)
a file for a pattern using full		regular expressions ..... egrep(1)
		reinit: run an initialization file ..... reinit(1F)
accept,		reject: allow or prevent LP requests ..... accept(1M)
comm: select or		reject lines common to two sorted files ..... comm(1)
join:		relational database operator ..... join(1)
t_sndrel: initiate an orderly		release ..... t_sndrel(3N)
acknowledge receipt of an orderly		release indication ..... t_rcvrel(3N)
common object file		reloc: relocation information for a ..... reloc(4)
ldrseek, ldnrseek: seek to		relocation entries of a section of a common/ ..... ldrseek(3X)
reloc:		relocation information for a common object file ..... reloc(4)
/ceil, fmod, fabs: floor, ceiling,		remainder, absolute value functions ..... floor(3M)
calendar:		reminder service ..... calendar(1)
uuxqt: execute		remote command requests ..... uuxqt(1M)
Uutry: try to contact		remote system with debugging on ..... uutry(1M)
ct: spawn getty to a		remote terminal ..... ct(1C)

rmdel: remove a delta from an SCCS file.....rmdel(1)
rmdir: remove a directory.....rmdir(2)
rmvopt: remove a menu option.....rmvopt(1M)
set, or shared memory ID ipcrm: remove a message queue, semaphore.....ipcrm(1)
unlink: remove directory entry.....unlink(2)
rm, rmdir: remove files or directories.....rm(1)
password entries pwfix: remove incorrectly formatted.....pwfix(1M)
deroff: remove nroff, tbl, and eqn constructs.....deroff(1)
rename: change the name of a file.....rename(2)
consistency check and interactive repair.....fsck(1M)
uniq: report repeated lines in a file.....uniq(1)
clock: report CPU time used.....clock(3C)
fsstat: report filesystem status.....fsstat(1M)
facilities status ipcs: report inter-process communication.....ipcs(1)
df: report number of free disk blocks and inodes.....df(1M)
mailbug: submit a bug report or enhancement request.....mailbug(1)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
timex: time a command; report process data and system activity.....timex(1)
ps: report process status.....ps(1)
uniq: report repeated lines in a file.....uniq(1)
fseek, rewind, ftell: reposition a file pointer in a stream.....fseek(3S)
filesystem repquota: summarize quotas for a.....repquota(1M)
submit a bug report or enhancement request.....mailbug(1)
t_accept: accept a connect request.....t_accept(3N)
t_listen: listen for a connect request.....t_listen(3N)
the confirmation from a connect request.....t_rcvconnect(3N)
send user-initiated disconnect request.....t_snddis(3N)
accept, reject: allow or prevent LP requests.....accept(1M)
the LP print service and move requests.....lpsched(1M)
uuxqt: execute remote command requests.....uuxqt(1M)
lp, cancel: send/cancel requests to the LP print service.....lp(1)
to its default values reset: reset the current form field.....reset(1F)
default values reset: reset the current form field to its.....reset(1F)
get_process_stats: get resource utilization info.....get_process_stats(2SEQ)
restore: incremental filesystem restore.....restore(1M)
restore restore: incremental filesystem.....restore(1M)
/ksh, rksh – KornShell, a standard/ restricted command and programming/.....ksh(1)
sh, rsh: shell, the standard/ restricted command programming/.....sh(1)
shadow: restricted password file.....shadow(4)
/m_rele_procs: suspend and resume child process execution.....m_park_procs(3PPS)
usage examples usage: retrieve a command description and.....usage(1)
t_rcvdis: retrieve information from disconnect.....t_rcvdis(3N)
object file symbol table/ ldgetname: retrieve symbol name for common.....ldgetname(3X)
abs: return integer absolute value.....abs(3C)
logname: return login name of user.....logname(3X)
m_get_numprocs: return number of child procs.....m_get_numprocs(3PPS)
m_get_myid: return process identification.....m_get_myid(3PPS)
getenv: return value for environment name.....getenv(3C)
stat: data returned by stat system call.....stat(5)
marked menu items getitems: returns a list of the currently.....getitems(1F)
getfrm: returns the current frame number.....getfrm(1F)
cpus_online: returns the number of CPUs on-line.....cpus_online(3PPS)

col: filter reverse line-feeds.....col(1)
ci: check in RCS revisions.....ci(1)
co: check out RCS revisions.....co(1)
rcsdiff: compare RCS revisions.....rcsdiff(1)
rcsmerge: merge RCS revisions.....rcsmerge(1)
pointer in a stream fseek, rewind, ftell: reposition a file.....fseek(3S)
/opendir, readdir, telldir, seekdir, rewinddir, closedir: directory/.....directory(3X)
creat: create a new file or rewrite an existing one.....creat(2)
restricted command/ ksh, rksh: KornShell, a standard/.....ksh(1)
information about RCS files rlog: print log messages and other.....rlog(1)
directories rm, rmdir: remove files or.....rm(1)
mail, rmail: send mail to users or read mail.....mail(1)
file rmdel: remove a delta from an SCCS.....rmdel(1)
rmdir: remove a directory.....rmdir(2)
rm, rmdir: remove files or directories.....rm(1)
rmvopt: remove a menu option.....rmvopt(1M)
chroot: change root directory.....chroot(2)
chroot: change root directory for a command.....chroot(1M)
logarithm, power, square root functions.....exp(3M)
queue menumsg: route user message to menu message.....menumsg(1M)
ldfcn: common object file access routines.....ldfcn(4)
regular expression compile and match routines.....regexp(5)
programming language sh, rsh: shell, standard/restricted command/.....sh(1)
nice: run a command at low priority.....nice(1)
nohup: run a command immune to hangups and quits.....nohup(1)
shell: run a command using shell.....shell(1F)
noage, nopff, noswap, onproc: run a command with priority aging, PFF/.....noage(1M)
the operating system boot: run a standalone program or bring up.....boot(8)
run: run an executable.....run(1F)
edc: run an externally described command.....edc(1M)
reinit: run an initialization file.....reinit(1F)
setpgrp: run command in its own process group.....setpgrp(1)
multi-user environment rc2: run commands performed for.....rc2(1M)
operating system rc0: run commands performed to stop the.....rc0(1M)
runacct: run daily accounting.....runacct(1M)
run: run an executable.....run(1F)
runacct: run daily accounting.....runacct(1M)
Administration SA: devices administered by System.....sa(7)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
editing activity sact: print current SCCS file.....sact(1)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
report package sar: sa1, sa2, sadc: system activity.....sar(1M)
sar: system activity reporter.....sar(1)
savecore: save a core dump of the operating system.....savecore(1M)
operating system savecore: save a core dump of the.....savecore(1M)
brk, sbrk: change data segment space allocation.....brk(2)
brk, sbrk: change private data segment size.....brk(3PPS)
formatted input scanf, fscanf, sscanf: convert.....scanf(3S)
bfs: big file scanner.....bfs(1)
awk: pattern scanning and processing language.....awk(1)
nawk: pattern scanning and processing language.....nawk(1)

change the delta commentary of an SCCS delta ..... cdc(1)
comb: combine SCCS deltas ..... comb(1)
delta: make a delta (change) to an SCCS file ..... delta(1)
get: get a version of an SCCS file ..... get(1)
prs: print an SCCS file ..... prs(1)
rmdel: remove a delta from an SCCS file ..... rmdel(1)
sccsdiff: compare two versions of an SCCS file ..... sccsdiff(1)
sccsfile: format of SCCS file ..... sccsfile(4)
sccstorcs: build RCS file from SCCS file ..... sccstorcs(1)
unget: undo a previous get of an SCCS file ..... unget(1)
val: validate SCCS file ..... val(1)
sact: print current SCCS file editing activity ..... sact(1)
admin: create and administer SCCS files ..... admin(1)
what: identify SCCS files ..... what(1)
SCCS file sccsdiff: compare two versions of an ..... sccsdiff(1)
sccsfile: format of SCCS file ..... sccsfile(4)
file sccstorcs: build RCS file from SCCS ..... sccstorcs(1)
board SCED: SCSI/Ethernet/Diagnostics ..... sced(7)
ckbupscd: check filesystem backup schedule ..... ckbupscd(1M)
transport program uusched: the scheduler for the uucp file ..... uusched(1M)
unlock locks s_init_lock, s_lock, s_clock, s_unlock: initialize, lock, ... s_lock(3PPS)
object file scnhdr: section header for a common ..... scnhdr(4)
image file scr_dump: format of curses screen ..... scr_dump(4)
more: view a file one full screen at a time ..... more(1)
curses: terminal screen handling and optimization package ..... curses(3X)
scr_dump: format of curses screen image file ..... scr_dump(4)
vi: screen-oriented (visual) display editor based on ex ..... vi(1)
inittab: script for the init process ..... inittab(4)
rc: reboot and shutdown command scripts ..... rc(4)
tg: SCSI 1/2-inch magnetic tape interface ..... tg(7)
sd: SCSI disk controller ..... sd(7)
wd: SSM SCSI disk controller ..... wd(7)
CCSformat: standalone embedded SCSI disk formatter ..... ccsformat(8)
tm: SCSI tape controller ..... tm(7)
ts: SCSI tape controller ..... ts(7)
SCED: SCSI/Ethernet/Diagnostics board ..... sced(7)
getscsiinfo, getscsimatch: get scsiinfo structure ..... getscsiinfo(3SEQ)
sd: SCSI disk controller ..... sd(7)
program sdiff: side-by-side difference ..... sdiff(1)
fgrep: search a file for a character string ..... fgrep(1)
grep: search a file for a pattern ..... grep(1)
full regular expressions egrep: search a file for a pattern using ..... egrep(1)
bsearch: binary search a sorted table ..... bsearch(3C)
acctcom: search and print process accounting files ..... acctcom(1)
lsearch, lfind: linear search and update ..... lsearch(3C)
pathconv: search Interpreter criteria for filename ..... pathconv(1F)
hcreate, hdestroy: manage hash search tables ..... hsearch(3C)
tfind, tdelete, twalk: manage binary search trees ..... tsearch(3C)
manipulate the object file comment section ..... mcs(1)
m_multi: mark single-process code section ..... m_single(3PPS)
scnhdr: section header for a common object file ..... scnhdr(4)
/ldnshread: read an indexednamed section header of a common object/ ..... ldshread(3X)

/seek to line number entries of a section of a common object file ..... ldlseek(3X)
/seek to relocation entries of a section of a common object file ..... ldrseek(3X)
/ldnsseek: seek to an indexednamed section of a common object file ..... ldsseek(3X)
dltsectbl: refresh security table ..... dltsectbl(1M)
task: global security table ..... task(4)
tasktbl: create security table in shared memory ..... tasktbl(1M)
sed: stream editor ..... sed(1)
/nrand48, mrand48, jrand48, srand48, seed48, lcong48: generate uniformly/ ..... drand48(3C)
a common object/ ldsseek, ldnsseek: seek to an indexednamed section of ..... ldsseek(3X)
section of a/ ldlseek, ldnlseek: seek to line number entries of a ..... ldlseek(3X)
ldrseek, ldnrseek: seek to relocation entries of a section of a/ ..... ldrseek(3X)
a common object file ldohseek: seek to the optional file header of ..... ldohseek(3X)
object file ldtbseek: seek to the symbol table of a common ..... ldtbseek(3X)
/opendir, readdir, telldir, seekdir, rewinddir, closedir/ ..... directory(3X)
sweepoff: enable/disable patrol seeks on DCC disks ..... sweepon(1M)
shmget: get shared memory segment identifier ..... shmget(2)
brk, sbrk: change private data segment size ..... brk(3PPS)
shbrk, shsbrk: change shared data segment size ..... shbrk(3PPS)
brk, sbrk: change data segment space allocation ..... brk(2)
sorted files comm: select or reject lines common to two ..... comm(1)
greek: select terminal filter ..... greek(1)
addformopt: add a form/ SELECT window option to a menu ..... addformopt(1M)
semctl: semaphore control operations ..... semctl(2)
semop: semaphore operations ..... semop(2)
ipcrm: remove a message queue, semaphore set, or shared memory ID ..... ipcrm(1)
semget: get set of semaphores ..... semget(2)
semctl: semaphore control operations ..... semctl(2)
semget: get set of semaphores ..... semget(2)
semop: semaphore operations ..... semop(2)
t_sndudata: send a data unit ..... t_sndudata(3N)
putmsg: send a message on a stream ..... putmsg(2)
kill: send a signal to a process or a group of processes ..... kill(2)
tfnotify: send completion message ..... tfnotify(1M)
connection t_snd: send data or expedited data over a ..... t_snd(3N)
mail, rmail: send mail to users or read mail ..... mail(1)
t_snddis: send user-initiated disconnect request ..... t_snddis(3N)
lp, cancel: send/cancel requests to the LP print service ..... lp(1)
receive data or expedited data sent over a connection ..... t_rcv(3N)
intro: introduction to Sequent system calls ..... intro(2SEQ)
dis386: disassemble Series 386 instructions ..... dis386(3SEQ)
SSM: System Services Module ..... ssm(7)
setsid: create session and set process group ID ..... setsid(2)
ascii: map of ASCII character set ..... ascii(5)
alarm: set a process alarm clock ..... alarm(2)
umask: set and get file creation mask ..... umask(2)
stty, gtty: set and get terminal state (defunct) ..... stty(2)
in core or in files set, unset: set and unset environment variables ..... set(1F)
sigstack: set and/or get signal stack context ..... sigstack(2SEQ)
timezone: set default system time zone ..... timezone(4)
backup.info: backup set definitions ..... backup_info(4)
setdtablesize: set descriptor table size ..... setdtablesize(2SEQ)
env: set environment for command execution ..... env(1)

utime: set file access and modification times.....utime(2)
umask: set file-creation mode mask.....umask(1)
tcsetpgrp: set foreground process group ID.....tcsetpgrp(2)
setgroups: set group access list.....setgroups(2SEQ)
cpset: copy file and set mode, user, and group IDs.....cpset(1)
m_set_procs: set number of child processes.....m_set_procs(3PPS)
semget: get set of semaphores.....semget(2)
remove a message queue, semaphore set, or shared memory ID.....ipcrm(1)
lpusers: set printing queue priorities.....lpusers(1M)
setpgrp: set process group ID.....setpgrp(2)
setsid: create session and set process group ID.....setsid(2)
setlocale: set programs location.....setlocale(3C)
tabs: set tabs on a terminal.....tabs(1)
tcgetattr, tcsetattr: get/ set terminal state.....tcgetattr(2)
line discipline getty: set terminal type, modes, speed, and.....getty(1M)
line discipline uugetty: set terminal type, modes, speed, and.....uugetty(1M)
date: print and set the date.....date(1)
stty: set the options for a terminal.....stty(1)
stime: set time.....stime(2)
environment variables in core or in/ set, unset: set and unset.....set(1F)
setuid, setgid: set user and group IDs.....setuid(2)
ulimit: get and set user limits.....ulimit(2)
a stream setbuf, setvbuf: assign buffering to.....setbuf(3S)
/toascci, _tolower, _toupper, setchrclass: character handling.....ctype(3C)
setcolor: redefine or create a color.....setcolor(1F)
size setdtablesize: set descriptor table.....setdtablesize(2SEQ)
setuid, setgid: set user and group IDs.....setuid(2)
getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent: get group/.....getgrent(3C)
setgroups: set group access list.....setgroups(2SEQ)
setjmp, longjmp: non-local goto.....setjmp(3C)
crypt, setkey, encrypt: generate hashing encryption.....crypt(3C)
setlocale: set programs location.....setlocale(3C)
setmnt: establish mount table.....setmnt(1M)
/endmntent, getmntent, hasmntopt, setmntent: get file system/.....getmntent(3SEQ)
setpgid: manipulate signal sets.....setpgid(2)
process group setpgrp: run command in its own.....setpgrp(1)
setpgrp: set process group ID.....setpgrp(2)
getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent: get/.....getpwent(3C)
process group ID setsid: create session and set.....setsid(2)
getspent, getspnam, setspent, endspent, fgetspent, lckpwdf,/.....getspent(3X)
profile: setting up an environment at login time.....profile(4)
gettydefs: speed and terminal settings used by getty.....gettydefs(4)
IDs setuid, setgid: set user and group.....setuid(2)
user setup: initialize system for first.....setup(1)
/getutid, getutline, pututline, setutent, endutent, utmpname,/.....getut(3C)
setbuf, setvbuf: assign buffering to a stream.....setbuf(3S)
machine-independent fashion sputl, sgetl: access long integer data in a.....sputl(3X)
standard/restricted command/ sh, rsh: shell, the.....sh(1)
/fgetspent, lckpwdf, ulckpwdf: get shadow password file entry.....getspent(3X)
putspent: write shadow password file entry.....putspent(3X)
shadow: restricted password file.....shadow(4)
pwconv: installs and updates /etc/ shadow with information from/.....pwconv(1M)

shbrk, shsbrk: change — shared data segment size ..... shbrk(3PPS)
tasktbl: create security table in — shared memory ..... tasktbl(1M)
shmalloc, shrealloc, shfree: — shared memory allocator ..... shmalloc(3PPS)
shmctl: — shared memory control operations ..... shmctl(2)
a message queue, semaphore set, or — shared memory ID ..... ipcrm(1)
shmop, shmat, shmdt: — shared memory operations ..... shmop(2)
shmget: get — shared memory segment identifier ..... shmget(2)
segment size — shbrk, shsbrk: change shared data ..... shbrk(3PPS)
shell: run a command using — shell ..... shell(1F)
assign menu system as user login — shell ..... tfmenush(1M)
system: issue a — shell command ..... system(3S)
csh: a — shell (command interpreter) with C-like syntax ..... csh(1)
/shutacct, startup, turnacct: — shell procedures for accounting ..... acctsh(1M)
shell: run a command using shell ..... shell(1F)
shells: — shell table ..... shells(4)
command programming/ sh, rsh: — shell, the standard/restricted ..... sh(1)
shells: shell table ..... shells(4)
shmalloc, shrealloc, — shfree: shared memory allocator ..... shmalloc(3PPS)
memory allocator — shmalloc, shrealloc, shfree: shared ..... shmalloc(3PPS)
shmop, — shmat, shmdt: shared memory operations ..... shmop(2)
operations — shmctl: shared memory control ..... shmctl(2)
shmop, shmat, — shmdt: shared memory operations ..... shmop(2)
identifier — shmget: get shared memory segment ..... shmget(2)
operations — shmop, shmat, shmdt: shared memory ..... shmop(2)
the machine — showcfg: show the configuration of ..... showcfg(1M)
allocator shmalloc, — shrealloc, shfree: shared memory ..... shmalloc(3PPS)
shbrk, — shsbrk: change shared data segment size ..... shbrk(3PPS)
shutdown: — shut down system, change system state ..... shutdown(1M)
/nulladm, prctmp, prdaily, prtacct, — shutacct, startup, turnacct: shell/ ..... acctsh(1M)
rc: reboot and — shutdown command scripts ..... rc(4)
shutdown: system — shutdown information file ..... shutdown(4)
system state — shutdown: shut down system, change ..... shutdown(1M)
information file — shutdown: system shutdown ..... shutdown(4)
actions — sigaction: examine and change signal ..... sigaction(2)
manipulate/ sigemptyset, sigfillset, — sigaddset, sigdelset, sigismember: ..... sigemptyset(2)
sigcontext: get signal context ..... sigcontext(2SEQ)
sigemptyset, sigfillset, sigaddset, — sigdelset, sigismember: manipulate/ ..... sigemptyset(2)
sigdelset, sigismember: manipulate/ — sigemptyset, sigfillset, sigaddset, ... sigemptyset(2)
sigismember:/ sigemptyset, — sigfillset, sigaddset, sigdelset, ... sigemptyset(2)
sigpause: signal management sigset, — sighold, sigrelse, sigignore, ... sigset(2)
sigset, sighold, sigrelse, — sigignore, sigpause: signal/ ..... sigset(2)
/sigfillset, sigaddset, sigdelset, — sigismember: manipulate signal sets ..... sigemptyset(2)
sigsetjmp, — siglongjmp: non-local jumps ..... sigsetjmp(3C)
login: — sign on ..... login(1)
pause: suspend process until — signal ..... pause(2)
specify what to do upon receipt of a — signal ..... signal(2)
sigsuspend: wait for a — signal ..... sigsuspend(2)
sigaction: examine and change — signal actions ..... sigaction(2)
sigcontext: get — signal context ..... sigcontext(2SEQ)
sigrelse, sigignore, sigpause: — signal management ..... sigset(2)
setpgid: manipulate — signal sets ..... setpgid(2)
sigdelset, sigismember: manipulate — signal sets ..... sigemptyset(2)

receipt of a signal — signal: specify what to do upon ..... signal(2)
sigstack: set and/or get — signal stack context ..... sigstack(2SEQ)
kill: send a — signal to a process or a group of processes ..... kill(2)
sigpending: examine pending — signals ..... sigpending(2)
examine and change blocked — signals ..... sigprocmask(2)
ssignal, gsignal: software — signals ..... ssignal(3C)
sighold, sigrelse, sigignore, — sigpause: signal management ..... sigset(2)
— sigpending: examine pending signals ..... sigpending(2)
blocked signals — sigprocmask: examine and change ..... sigprocmask(2)
signal management sigset, sighold, — sigrelse, sigignore, sigpause: ..... sigset(2)
sigignore, sigpause: signal/ — sigset, sighold, sigrelse, ... sigset(2)
jumps — sigsetjmp, siglongjmp: non-local ..... sigsetjmp(3C)
stack context — sigstack: set and/or get signal ..... sigstack(2SEQ)
— sigsuspend: wait for a signal ..... sigsuspend(2)
atan2: trigonometric/ trig: — sin, cos, tan, asin, acos, atan, ... trig(3M)
m_single, m_multi: mark — single-process code section ..... m_single(3PPS)
functions — sinh, cosh, tanh: hyperbolic ..... sinh(3M)
initialize barrier, wait at barrier — s_init_barrier, s_wait_barrier: ..... s_wait_barrier(3PPS)
s_unlock: initialize, lock, unlock/ — s_init_lock, s_lock, s_clock, ... s_lock(3PPS)
sbrk: change private data segment — size ..... brk(3PPS)
getdtablesize: get descriptor table — size ..... getdtablesize(2SEQ)
getpagesize: get system page — size ..... getpagesize(2SEQ)
setdtablesize: set descriptor table — size ..... setdtablesize(2SEQ)
shsbrk: change shared data segment — size ..... shbrk(3PPS)
of object files — size: print section sizes ..... see *C and Lang Tools Binder*
interval — sleep: suspend execution for an ..... sleep(1)
interval — sleep: suspend execution for ..... sleep(3C)
s_init_lock, — s_lock, s_clock, s_unlock: initialize, lock,/ ..... s_lock(3PPS)
ttyslot: find the — slot in the utmp file of the current user ..... ttyslot(3C)
consmem, — sm, ss: console memory drivers ..... consmem(7)
ssignal, gsignal: — software signals ..... ssignal(3C)
qsort: quicker — sort ..... qsort(3C)
tsort: topological — sort ..... tsort(1)
sort: — sort and/or merge files ..... sort(1)
— sort: sort and/or merge files ..... sort(1)
select or reject lines common to two — sorted files ..... comm(1)
bsearch: binary search a — sorted table ..... bsearch(3C)
an object file list: produce C — source listing from ..... see *C and Lang Tools Binder*
— sp: STREAMS pipe device ..... sp(7)
open file into the processs address — space ..... mmap(2SEQ)
unmap part of the processs address — space ..... munmap(2SEQ)
into processs virtual address — space ..... pmap(7)
brk, sbrk: change data segment — space allocation ..... brk(2)
ct: — spawn getty to a remote terminal ..... ct(1C)
memory efficient way vfork: — spawn new process in a virtual ..... vfork(2)
fspec: format — specification in text files ..... fspec(4)
signal: — specify what to do upon receipt of a signal ..... signal(2)
getty: set terminal type, modes, — speed, and line discipline ..... getty(1M)
uugetty: set terminal type, modes, — speed, and line discipline ..... uugetty(1M)
gettydefs: — speed and terminal settings used by getty ..... gettydefs(4)
find spelling errors — spell, hashmake, spellin, hashcheck: ..... spell(1)
file — spellhist: unmatched spelling words ..... spellhist(4)

spell, hashmake, hashmake, spellin, hashcheck: find — spellin, hashcheck: find spelling errors ..... spell(1)
spell, hashmake, hashmake, spellin, hashcheck: find — spelling errors ..... spell(1)
spellhist: unmatched — spelling words file ..... spellhist(4)
csplit: context — split ..... csplit(1)
split: — split a file into pieces ..... split(1)
fsplit: — split f77, ratfor, or efl files ..... fsplit(1)
— split: split a file into pieces ..... split(1)
uucleanup: uucp — spool directory clean-up ..... uucleanup(1M)
printf, fprintf, — sprintf: print formatted output ..... printf(3S)
data in a machine-independent/ — sputl, sgetl: access long integer ..... sputl(3X)
square root/ exp, log, log10, pow, — sqrt: exponential, logarithm, power, ... exp(3M)
sqrt: exponential, logarithm, power, — square root functions ..... exp(3M)
rand, — srand: simple random-number generator ..... rand(3C)
/lrand48, nrand48, mrand48, jrand48, — srand48, seed48, lcong48: generate/ ..... drand48(3C)
consmem, sm, — ss: console memory drivers ..... consmem(7)
scanf, fscanf, — sscanf: convert formatted input ..... scanf(3S)
— ssignal, gsignal: software signals ..... ssignal(3C)
wp: — SSM parallel line printer interface ..... wp(7)
wd: — SSM SCSI disk controller ..... wd(7)
— SSM: System Services Module ..... ssm(7)
terminal multiplexor driver — st: Systech MTI-800/1600/1650 ..... st(7)
sigstack: set and/or get signal — stack context ..... sigstack(2SEQ)
copy: perform a — standalone data transfer ..... copy(8)
prtvtoc: print the VTOC of a — standalone disk device ..... prtvtoc(8)
dual-channel disk/ zdformat: — standalone disk formatter for ..... zdformat(8)
CCSformat: — standalone embedded SCSI disk formatter ..... ccsformat(8)
dump: — standalone memory dump ..... dump(8)
operating system boot: run a — standalone program or bring up the ..... boot(8)
intro: introduction to system — standalone utilities ..... intro(8)
stdio: — standard buffered input/output package ..... stdio(3S)
package stdipc: ftok: — standard interprocess communication ..... stdipc(3C)
sh, rsh: shell, the — standard/restricted command programming/ ..... sh(1)
operating system for beginning/ — starter: information about the ..... starter(1)
lpsched, lpshut, lpmove: — start/stop the LP print service and move/ ..... lpsched(1M)
/prctmp, prdaily, prtacct, shutacct, — startup, turnacct: shell procedures/ ..... acctsh(1M)
call — stat: data returned by stat system ..... stat(5)
— stat, lstat, fstat: get file status ..... stat(2)
stat: data returned by — stat system call ..... stat(5)
information — statfs, fstatfs: get filesystem ..... statfs(2)
ustat: get filesystem — statistics ..... ustat(2)
ff: list filenames and — statistics for a filesystem ..... ff(1M)
fsstat: report filesystem — status ..... fsstat(1M)
communication facilities — status ..... ipcs(1)
ps: report process — status ..... ps(1)
stat, lstat, fstat: get file — status ..... stat(2)
feof, clearerr, fileno: stream — status inquiries ..... ferror(3S)
uustat: uucp — status inquiry and job control ..... uustat(1C)
lpstat: print information about the — status of the LP print service ..... lpstat(1)
input/output package — stdio: standard buffered ..... stdio(3S)
communication package — stdipc: ftok: standard interprocess ..... stdipc(3C)
— stime: set time ..... stime(2)
waitpid: wait for a child process to — stop or terminate ..... wait(2)

lpsched, lpshut, lpmove: start/ stop the LP print service and move/.....lpsched(1M)
rc0: run commands performed to stop the operating system.....rc0(1M)
icheck: filesystem storage consistency check.....icheck(1M)
strace: print STREAMS trace messages.....strace(1M)
strncmp, strcpy, strncpy,/ string: strcat, strdup, strncat, strcmp,.....string(3C)
/strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn,/.....string(3C)
cleanup program strclean: STREAMS error logger.....strclean(1M)
string: strcat, strdup, strncat, strncmp, strcpy, strncpy, strlen, strchr,/.....string(3C)
/strdup, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr,/.....string(3C)
/strchr, strrchr, strpbrk, strspn, strcspn, strtok: string operations.....string(3C)
strcpy, strncpy,/ string: strcat, strdup, strncat, strcmp, strncmp,.....string(3C)
fclose, fflush: close or flush a stream.....fclose(3S)
fopen, freopen, fdopen: open a stream.....fopen(3S)
reposition a file pointer in a stream.....fseek(3S)
getw: get character or word from a stream.....getc(3S)
getmsg: get next message off a stream.....getmsg(2)
gets, fgets: get a string from a stream.....gets(3S)
putw: put character or word on a stream.....putc(3S)
putmsg: send a message on a stream.....putmsg(2)
puts, fputs: put a string on a stream.....puts(3S)
setvbuf: assign buffering to a stream.....setbuf(3S)
push character back into input stream.....ungetc(3S)
sed: stream editor.....sed(1)
ferror, feof, clearerr, fileno: stream status inquiries.....ferror(3S)
streamio: STREAMS ioctl commands.....streamio(7)
driver streampty: STREAMS pseudo terminal .....streampty(7)
clone: open any minor device on a STREAMS driver.....clone(7)
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)
poll: STREAMS input/output multiplexing.....poll(2)
streamio: STREAMS ioctl commands.....streamio(7)
Transport Interface cooperating STREAMS module.....timod(7)
Interface read/write interface STREAMS module.....tirdwr(7)
sp: STREAMS pipe device.....sp(7)
streampty: STREAMS pseudo terminal driver.....streampty(7)
strace: print STREAMS trace messages.....strace(1M)
streamtty: Generic STREAMS tty driver interface.....streamtty(7)
ld0: STREAMS tty line discipline .....ld0(7)
driver interface streamtty: Generic STREAMS tty .....streamtty(7)
strerr: STREAMS error logger daemon .....strerr(1M)
strftime: format time information.....strftime(3C)
long integer and base-64 ASCII string.....a64l(3C)
tzset: convert date and time to string.....ctime(3C)
convert floating-point number to string.....ecvt(3C)
fgrep: search a file for a character string.....fgrep(1)
regex: match patterns against a string.....regex(1F)
gps: graphical primitive string, format of graphical files.....gps(4)
gets, fgets: get a string from a stream.....gets(3S)
puts, fputs: put a string on a stream.....puts(3S)
echo: put string on virtual output .....echo(1F)
strpbrk, strspn, strcspn, strtok: string operations.....string(3C)

strcmp, strncmp, strcpy, strncpy,/ string: strcat, strdup, strncat, ...string(3C)
strtod, atof: convert string to double-precision number.....strtod(3C)
strtol, atol, atoi: convert string to integer.....strtol(3C)
cftime: language specific strings .....cftime(4)
object file strip: strip an.....see *C and Lang Tools Binder*
/strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk,/.....string(3C)
string: strcat, strdup, strncat, strcmp, strncmp, strcpy, strncpy,/.....string(3C)
/strcat, strdup, strncat, strcmp, strncmp, strcpy, strlen,/.....string(3C)
/strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr,/.....string(3C)
/strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok:/.....string(3C)
/strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn,/.....string(3C)
/strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok: string/.....string(3C)
character strstr: find first occurrence of.....strstr(3C)
double-precision number strtod, atof: convert string to.....strtod(3C)
strrchr, strpbrk, strspn, strcspn, strtok: string operations .....string(3C)
to integer strtol, atol, atoi: convert string .....strtol(3C)
processes using a file or file structure.....fuser(1M)
getscsimatch: get scsiinfo structure.....getscsiinfo(3SEQ)
getzdinfobyname: get zdinfo structure .....getzdinfo(3SEQ)
read constab entries into a structure .....read_constab(3SEQ)
t_alloc: allocate a library structure.....t_alloc(3N)
t_free: free a library structure.....t_free(3N)
state (defunct) stty, gtty: set and get terminal .....stty(2)
stty: set the options for a terminal.....stty(1)
su: become superuser or another user.....su(1M)
backup.codes1: backup subcode definitions.....backup_codes1(4)
m_fork: execute a subprogram in parallel.....m_fork(3PPS)
dbm_clearerr, dmb_blkflush: database subroutines .....ndbm(3X)
plot: graphics interface subroutines.....plot(3X)
sulog: log of substitute user events.....sulog(4)
sulog: log of substitute user events.....sulog(4)
of a file sum: print checksum and block count.....sum(1)
du: summarize disk usage.....du(1M)
repquota: summarize quotas for a filesystem .....repquota(1M)
records acctcms: command summary from per-process accounting .....acctcms(1M)
s_init_lock, s_lock, s_clock, s_unlock: initialize, lock, unlock locks.....s_lock(3PPS)
sync: update the superblock .....sync(1M)
sync: update superblock.....sync(2)
su: become superuser or another user .....su(1M)
getgroups: get supplementary group IDs.....getgroups(2)
uucp: networking support files.....uucp(4)
lptypes: table of printer types supported by the menu system.....lptypes(4)
m_park_procs, m_rele_procs: suspend and resume child process/.....m_park_procs(3PPS)
sleep: suspend execution for an interval.....sleep(1)
sleep: suspend execution for interval.....sleep(3C)
pause: suspend process until signal.....pause(2)
swab: swap bytes.....swab(3C)
wait at barrier s_init_barrier, s_wait_barrier: initialize barrier,.....s_wait_barrier(3PPS)
swap: swap administrative interface.....swap(1M)
swab: swap bytes.....swab(3C)
swap: swap administrative interface .....swap(1M)
/a command with priority aging, PFF, swapping, or processor migration/.....noage(1M)

seeks on DCC disks sweepon, sweepoff: enable/disable patrol ..... sweepon(1M)
patrol seeks on DCC disks sweepon, sweepoff: enable/disable ..... sweepon(1M)
symbol table/ ldgetname: retrieve symbol name for common object file ..... ldgetname(3X)
symbol name for common object file ..... ldgetname(3X)
ldtbindex: compute the index of a symbol table entry of a common/ ..... ldtbindex(3X)
ldtbread: read an indexed symbol table entry of a common object/ ..... ldtbread(3X)
syms: common object file symbol table format ..... syms(4)
ldtbseek: seek to the symbol table of a common object file ..... ldtbseek(3X)
unistd: file header for symbolic constants ..... unistd(4)
readlink: read value of a symbolic link ..... readlink(2SEQ)
symlink: make symbolic link to a file ..... symlink(2SEQ)
of common operating system terms and symbols ..... glossary(1)
file symlink: make symbolic link to a ..... symlink(2SEQ)
table format syms: common object file symbol ..... syms(4)
sync: update superblock ..... sync(2)
sync: update the superblock ..... sync(1M)
controlling FMLI object vsig: synchronize a co-process with the ..... vsig(1F)
t_sync: synchronize transport library ..... t_sync(3N)
check documents for correct eqn syntax ..... checkeq(1)
(command interpreter) with C-like syntax ..... csh(1)
administration sysadm: menu interface to do system ..... sysadm(1)
syscall: indirect system call ..... syscall(2)
variables sysconf: get configurable system ..... sysconf(2)
configuration file sys_config: format of a system ..... sys_config(4)
perror, errno, sys_errlist, sys_nerr: system error messages ..... perror(3C)
sysi86: machine specific functions ..... sysi86(2)
menu system functions syskeymap: identify keys to internal ..... syskeymap(4)
perror, errno, sys_errlist, sys_nerr: system error messages ..... perror(3C)
multiplexor driver st: Systech MTI-800/1600/1650 terminal ..... st(7)
lp: Systech parallel line printer interface ..... lp(7)
bsearch: binary search a sorted table ..... bsearch(3C)
constab: constructed devices table ..... constab(4)
devtab: virtual device table ..... devtab(4)
dltsectbl: refresh security table ..... dltsectbl(1M)
kernel auto-configuration device table ..... dumpconf(1M)
fstab: file-system- table ..... fstab(4)
mnttab: mounted filesystem table ..... mnttab(4)
setmnt: establish mount table ..... setmnt(1M)
shells: shell table ..... shells(4)
task: global security table ..... task(4)
name for common object file symbol table entry ..... ldgetname(3X)
/compute the index of a symbol table entry of a common object file ..... ldtbindex(3X)
ldtbread: read an indexed symbol table entry of a common object file ..... ldtbread(3X)
syms: common object file symbol table format ..... syms(4)
tasktbl: create security table in shared memory ..... tasktbl(1M)
get maximum in-use descriptor table index ..... getdtablemax(2SEQ)
ldtbseek: seek to the symbol table of a common object file ..... ldtbseek(3X)
the menu system lptypes: table of printer types supported by ..... lptypes(4)
getdtablesize: get descriptor table size ..... getdtablesize(2SEQ)
setdtablesize: set descriptor table size ..... setdtablesize(2SEQ)
classification and conversion tables ..... chrtbl(1M)
hdestroy: manage hash search tables ..... hsearch(3C)

tabs: set — tabs on a terminal ..... tabs(1)
tabs: set tabs on a terminal ..... tabs(1)
t_accept: accept a connect request ..... t_accept(3N)
ctags: create a — tags file ..... ctags(1)
file — tail: deliver the last part of a ..... tail(1)
structure — t_alloc: allocate a library ..... t_alloc(3N)
trig: sin, cos, — tan, asin, acos, atan, atan2: trigonometric/ ..... trig(3M)
sinh, cosh, — tanh: hyperbolic functions ..... sinh(3M)
tm: SCSI — tape controller ..... tm(7)
ts: SCSI — tape controller ..... ts(7)
hpio: Hewlett-Packard 2645A terminal — tape file archiver ..... hpio(1)
tar: — tape file archiver ..... tar(1)
mtio: UNIX magnetic — tape interface ..... mtio(7)
tg: SCSI 1/2-inch magnetic — tape interface ..... tg(7)
xt: Xylogics 472 MULITBUS magnetic — tape interface ..... xt(7)
mt: magnetic — tape manipulating program ..... mt(1)
tar: tape file archiver ..... tar(1)
task: global security table ..... task(4)
shared memory — tasktbl: create security table in ..... tasktbl(1M)
transport endpoint — t_bind: bind an address to a ..... t_bind(3N)
deroff: remove nroff, — tbl, and eqn constructs ..... deroff(1)
terminal state — tcgetattr, tcsetattr: get/set ..... tcgetattr(2)
group ID — tcgetpgrp: get foreground process ..... tcgetpgrp(2)
t_close: close a transport endpoint ..... t_close(3N)
with another transport user — t_connect: establish a connection ..... t_connect(3N)
tcgetattr, — tcsetattr: get/set terminal state ..... tcgetattr(2)
group ID — tcsetpgrp: set foreground process ..... tcsetpgrp(2)
tsearch, tfind, — tdelete, twalk: manage binary search trees ..... tsearch(3C)
tee: pipe fitting ..... tee(1)
4014: paginator for the — Tektronix 4014 terminal ..... 4014(1)
init, — telinit: process control initialization ..... init(1M)
directory: opendir, readdir, — telldir, seekdir, rewinddir/ ..... directory(3X)
temporary file tmpnam, — tempnam: create a name for a ..... tmpnam(3S)
tmpfile: create a — temporary file ..... tmpfile(3S)
tmpnam, tempnam: create a name for a — temporary file ..... tmpnam(3S)
terminals — term: conventional names for ..... term(5)
term: format of compiled — term file ..... term(4)
term: format of compiled term file ..... term(4)
description captoinfo: convert a — termcap description into a terminfo ..... captoinfo(1M)
base — termcap: terminal capability data ..... termcap(4)
paginator for the Tektronix 4014 — terminal ..... 4014(1)
special functions of the DASI 450 — terminal ..... 450(1)
ct: spawn getty to a remote — terminal ..... ct(1C)
ctermid: generate filename for — terminal ..... ctermid(3S)
hangup the specified controlling — terminal ..... fvhangup(2SEQ)
stty: set the options for a — terminal ..... stty(1)
tabs: set tabs on a — terminal ..... tabs(1)
tty: get the name of the — terminal ..... tty(1)
ttyname, isatty: find name of a — terminal ..... ttyname(3C)
/associated with a terminal or the — terminal associated with a device ..... ttydevname(1)
termcap: — terminal capability data base ..... termcap(4)
terminfo: — terminal capability database ..... terminfo(4)

streampty: STREAMS pseudo terminal driver.....streampty(7)
greek: select terminal filter.....greek(1)
tty: controlling terminal (gentty) interface.....tty(7)
termio: general terminal interface.....termio(7)
dial: establish an out-going terminal line connection.....dial(3C)
st: Systech MTI-800/1600/1650 terminal multiplexor driver.....st(7)
xs: Xylogics 781 terminal multiplexor driver.....xs(7)
tput: initialize a terminal or query terminfo database.....tput(1)
/get the device associated with a terminal or the terminal associated/.....ttydevname(1)
optimization package curses: terminal screen handling and.....curses(3X)
gettydefs: speed and terminal settings used by getty.....gettydefs(4)
tcgetattr, tcsetattr: get/set terminal state.....tcgetattr(2)
stty, gtty: set and get terminal state (defunct).....stty(2)
hpio: Hewlett-Packard 2645A terminal tape file archiver.....hpio(1)
line discipline getty: set terminal type, modes, speed, and.....getty(1M)
line discipline uugetty: set terminal type, modes, speed, and.....uugetty(1M)
functions of DASI 300 and 300s terminals.....300(1)
getpseudotty: get pseudo terminals.....getpseudotty(3SEQ)
special functions of Hewlett-Packard terminals.....hp(1)
term: conventional names for terminals.....term(5)
wait for a child process to stop or terminate.....wait(2)
kill: terminate a process.....kill(1)
exit, _exit: terminate process.....exit(2)
tic: terminfo compiler.....tic(1M)
tput: initialize a terminal or query terminfo database.....tput(1)
convert a termcap description into a terminfo description.....captoinfo(1M)
infocmp: compare or print out terminfo descriptions.....infocmp(1M)
database terminfo: terminal capability.....terminfo(4)
termio: general terminal interface.....termio(7)
of common operating system terms and symbols.....glossary(1)
t_error: produce error message.....t_error(3N)
test: condition evaluation command.....test(1)
isnan: isnand, isnanf: test for floating-point NaN/.....isnan(3C)
ed, red: text editor.....ed(1)
ex: text editor.....ex(1)
edit: text editor (variant of ex for casual users).....edit(1)
errormsg: error message text file.....errormsg(4)
newform: change the format of a text file.....newform(1)
fspec: format specification in text files.....fspec(4)
isismgr: ISIS file manager, updates text files in ISIS file format.....isismgr(1M)
plock: lock process, text, or data in memory.....plock(2)
search trees tsearch, tfind, tdelete, twalk: manage binary.....tsearch(3C)
reporter tfipc: interprocess communication.....tfipc(1M)
login shell tfmenush: assign menu system as user.....tfmenush(1M)
tfnotify: send completion message.....tfnotify(1M)
t_free: free a library structure.....t_free(3N)
interface tg: SCSI 1/2-inch magnetic tape.....tg(7)
service information t_getinfo: get protocol-specific.....t_getinfo(3N)
t_getstate: get the current state.....t_getstate(3N)
merge: three-way file merge.....merge(1)
tic: terminfo compiler.....tic(1M)
batch: execute commands at a later time.....at(1)

view a file one full screen at a time ..... more(1)
setting up an environment at login time ..... profile(4)
stime: set time ..... stime(2)
time: get time ..... time(2)
time, ptime: time a command ..... time(1)
and system activity timex: time a command; report process data ..... timex(1)
acctcon1, acctcon2: connect- time accounting ..... acctcon(1M)
time: get time ..... time(2)
strftime: format time information ..... strftime(3C)
profil: execution time profile ..... profil(2)
time, ptime: time a command ..... time(1)
mktime, tzset: convert date and time to string ..... ctime(3C)
clock: report CPU time used ..... clock(3C)
timezone: set default system time zone ..... timezone(4)
times: get process and child process times ..... times(2)
set file access and modification times ..... utime(2)
times: get process and child process ..... times(2)
update access and modification times of a file ..... touch(1)
process data and system activity timex: time a command; report ..... timex(1)
zone timezone: set default system time ..... timezone(4)
cooperating STREAMS module timod: Transport Interface ..... timod(7)
read/write interface STREAMS module tirdwr: Transport Interface ..... tirdwr(7)
request t_listen: listen for a connect ..... t_listen(3N)
a transport endpoint t_look: look at the current event on ..... t_look(3N)
tm: SCSI tape controller ..... tm(7)
processor tmp_affinity: bind a process to a ..... tmp_affinity(2SEQ)
tmp_ctl: multi-processor control ..... tmp_ctl(2SEQ)
tmpfile: create a temporary file ..... tmpfile(3S)
temporary file tmpnam, tempnam: create a name for a ..... tmpnam(3S)
/isgraph, isascii, tolower, toupper, toascci, _tolower, _toupper/ ..... ctype(3C)
/tolower, _toupper, _tolower, toascii: translate characters ..... conv(3C)
conv: toupper, tolower, _toupper, _tolower, toascii: translate/ ..... conv(3C)
/isascii, tolower, toupper, toascci, _tolower, _toupper, setchrclass:/ ..... ctype(3C)
/ispunct, isprint, isgraph, isascii, tolower, toupper, toascci, _tolower/ ..... ctype(3C)
conv: toupper, tolower, _toupper, _tolower, toascii: translate/ ..... conv(3C)
endpoint t_open: establish a transport ..... t_open(3N)
tsort: topological sort ..... tsort(1)
transport endpoint t_optmgmt: manage options for a ..... t_optmgmt(3N)
acctmerg: merge or add total accounting files ..... acctmerg(1M)
modification times of a file touch: update access and ..... touch(1)
/tolower, toupper, toascci, _tolower, _toupper, setchrclass: character/ ..... ctype(3C)
/isprint, isgraph, isascii, tolower, toupper, toascci, _tolower/ ..... ctype(3C)
conv: toupper, tolower, _toupper, _tolower, toascii: translate/ ..... conv(3C)
_tolower, toascii: translate/ conv: toupper, tolower, _toupper, ... conv(3C)
terminfo database tput: initialize a terminal or query ..... tput(1)
tr: translate characters ..... tr(1)
ptrace, mptrace: process trace facility ..... ptrace(2)
strace: print STREAMS trace messages ..... strace(1M)
to STREAMS error logging and event tracing ..... log(7)
copy: perform a standalone data transfer ..... copy(8)
init.d: system level transition files directory ..... init.d(4)
_toupper, _tolower, toascii: translate characters ..... conv(3C)

tr: translate characters ..... tr(1)
t_bind: bind an address to a transport endpoint ..... t_bind(3N)
t_close: close a transport endpoint ..... t_close(3N)
look at the current event on a transport endpoint ..... t_look(3N)
t_open: establish a transport endpoint ..... t_open(3N)
t_optmgmt: manage options for a transport endpoint ..... t_optmgmt(3N)
t_unbind: disable a transport endpoint ..... t_unbind(3N)
STREAMS module timod: Transport Interface cooperating ..... timod(7)
interface STREAMS module tirdwr: Transport Interface read/write ..... tirdwr(7)
t_sync: synchronize transport library ..... t_sync(3N)
the scheduler for the uucp file transport program ..... uusched(1M)
uucico: file transport program for the uucp system ..... uucico(1M)
establish a connection with another transport user ..... t_connect(3N)
data sent over a connection t_rcv: receive data or expedited ..... t_rcv(3N)
confirmation from a connect request t_rcvconnect: receive the ..... t_rcvconnect(3N)
disconnect t_rcvdis: retrieve information from ..... t_rcvdis(3N)
orderly release indication t_rcvrel: acknowledge receipt of an ..... t_rcvrel(3N)
t_rcvudata: receive a data unit ..... t_rcvudata(3N)
error indication t_rcvuderr: receive a unit data ..... t_rcvuderr(3N)
ftw: walk a file tree ..... ftw(3C)
tdelete, twalk: manage binary search trees ..... tsearch(3C)
atan, atan2: trigonometric/ trig: sin, cos, tan, asin, acos, ... trig(3M)
cos, tan, asin, acos, atan, atan2: trigonometric functions ..... trig(3M)
true, false: provide truth values ..... true(1)
truncate, ftruncate: truncate a file to a specified length ..... truncate(2SEQ)
to a specified length truncate, ftruncate: truncate a file ..... truncate(2SEQ)
u3b2, u3b5, vax: get processor type truth value ..... machid(1)
true, false: provide truth values ..... true(1)
ts: SCSI tape controller ..... ts(7)
manage binary search trees tsearch, tfind, tdelete, twalk: ..... tsearch(3C)
over a connection t_snd: send data or expedited data ..... t_snd(3N)
disconnect request t_snddis: send user-initiated ..... t_snddis(3N)
release t_sndrel: initiate an orderly ..... t_sndrel(3N)
t_sndudata: send a data unit ..... t_sndudata(3N)
tsort: topological sort ..... tsort(1)
library t_sync: synchronize transport ..... t_sync(3N)
interface tty: controlling terminal (gentty) ..... tty(7)
streamtty: Generic STREAMS tty driver interface ..... streamtty(7)
tty: get the name of the terminal ..... tty(1)
ld0: STREAMS tty line discipline ..... ld0(7)
between major/minor numbers and tty names ..... dev_to_ttyname(3C)
associated with a terminal or the/ ttydevname: get the device ..... ttydevname(1)
terminal ttyname, isatty: find name of a ..... ttyname(3C)
major/minor/ dev_to_ttyname, ttyname_to_dev: convert between ..... dev_to_ttyname(3C)
file of the current user ttyslot: find the slot in the utmp ..... ttyslot(3C)
endpoint t_unbind: disable a transport ..... t_unbind(3N)
parameters vmtune: tune system-wide virtual-memory ..... vmtune(1M)
tunefs: tune up an existing filesystem ..... tunefs(1M)
filesystem tunefs: tune up an existing ..... tunefs(1M)
examine and change virtual memory tuning parameters ..... vm_ctl(2SEQ)
/prdaily, prtacct, shutacct, startup, turnacct: shell procedures for/ ..... acctsh(1M)
tsearch, tfind, tdelete, twalk: manage binary search trees ..... tsearch(3C)

types: primitive system data — types.....types(5)
types: primitive system data types.....types(5)
lptypes: table of printer — types supported by the menu system.....lptypes(4)
/asctime, cftime, ascftime, mktime, — tzset: convert date and time to/.....ctime(3C)
machid: i386, ns32000, pdp11, — u3b, u3b2, u3b5, vax: get processor type/.....machid(1)
machid: i386, ns32000, pdp11, u3b, — u3b2, u3b5, vax: get processor type/.....machid(1)
/i386, ns32000, pdp11, u3b, u3b2, — u3b5, vax: get processor type truth/.....machid(1)
uadmin: administrative control.....uadmin(1M)
uadmin: administrative control.....uadmin(2)
/endspent, fgetspent, lckpwdf, — ulckpwdf: get shadow password file entry.....getspent(3X)
ulimit: get and set user limits.....ulimit(2)
mask — umask: set and get file creation.....umask(2)
umask: set file-creation mode mask.....umask(1)
mount, — umount: mount and unmount filesystems.....mount(1M)
umount: unmount a filesystem.....umount(2)
filesystems — mountall, — umountall: mount, unmount multiple.....mountall(1M)
system — uname: get name of current operating.....uname(2)
uname: print name of current system.....uname(1)
expand data — compress, — uncompress, zcat: compress and.....compress(1)
unget: — undo a previous get of an SCCS file.....unget(1)
SCCS file — unget: undo a previous get of an.....unget(1)
input stream — ungetc: push character back into.....ungetc(3S)
file — uniq: report repeated lines in a.....uniq(1)
mktemp: make a — unique filename.....mktemp(3C)
constants — unistd: file header for symbolic.....unistd(4)
t_rcvudata: receive a data — unit.....t_rcvudata(3N)
t_sndudata: send a data — unit.....t_sndudata(3N)
t_rcvuderr: receive a — unit data error indication.....t_rcvuderr(3N)
units: conversion program.....units(1)
mtio: — UNIX magnetic tape interface.....mtio(7)
cu: call another — UNIX system.....cu(1C)
link, unlink: link and — unlink files and directories.....link(1M)
link, — unlink: link and unlink files and directories.....link(1M)
unlink: remove directory entry.....unlink(2)
m_lock, m_unlock: lock, — unlock locks.....m_lock(3PPS)
s_clock, s_unlock: initialize, lock, — unlock locks.....s_lock(3PPS)
space — munmap: — unmap part of the processs address.....munmap(2SEQ)
spellhist: — unmatched spelling words file.....spellhist(4)
umount: — unmount a filesystem.....umount(2)
mount, umount: mount and — unmount filesystems.....mount(1M)
mountall, umountall: mount, — unmount multiple filesystems.....mountall(1M)
pack, pcat, — unpack: compress and expand files.....pack(1)
set, unset: set and — unset environment variables in core or in files.....set(1F)
variables in core or in files — set, — unset: set and unset environment.....set(1F)
lsearch, lfind: linear search and — update.....lsearch(3C)
touch: — update access and modification times of a file.....touch(1)
make: maintain, — update, and regenerate groups of programs.....make(1)
bdflush: — update daemon.....bdflush(1M)
bdflush: — update daemon.....bdflush(2SEQ)
sync: — update superblock.....sync(2)
sync: — update the superblock.....sync(1M)
from/ — pwconv: installs and — updates /etc/shadow with information.....pwconv(1M)

isismgr: ISIS file manager, updates text files in ISIS file format ..... isismgr(1M)
been up uptime: show how long system has ..... uptime(1)
du: summarize disk usage ..... du(1M)
quota: display disk usage and limits ..... quota(1)
retrieve a command description and usage examples ..... usage(1)
description and usage examples usage: retrieve a command ..... usage(1)
usclk: 32-bit microsecond counter ..... usclk(7)
getusclk, usclk_init: get/init microsecond clock ..... getusclk(3SEQ)
get character login name of the user ..... cuserid(3S)
logname: return login name of user ..... logname(3X)
setup: initialize system for first user ..... setup(1)
su: become superuser or another user ..... su(1M)
a connection with another transport user ..... t_connect(3N)
slot in the utmp file of the current user ..... ttyslot(3C)
write: write to another user ..... write(1)
cpset: copy file and set mode, user, and group IDs ..... cpset(1)
setuid, setgid: set user and group IDs ..... setuid(2)
id: print user and group IDs and names ..... id(1M)
crontab: user crontab file ..... crontab(1)
addusr.def: add user defaults ..... addusr_def(4)
/geteuid, getgid, getegid: get real user, effective user, real group,/ ..... getuid(2)
environ: user environment ..... environ(5)
run commands performed for multi- user environment ..... rc2(1M)
sulog: log of substitute user events ..... sulog(4)
generate disk accounting data by user ID ..... diskusg(1M)
ulimit: get and set user limits ..... ulimit(2)
tfmenush: assign menu system as user login shell ..... tfmenush(1M)
menumsg: route user message to menu message queue ..... menumsg(1M)
edquota: edit user quotas ..... edquota(1M)
/getegid: get real user, effective user, real group, and effective/ ..... getuid(2)
t_snddis: send user-initiated disconnect request ..... t_snddis(3N)
editor (variant of ex for casual users) ..... edit(1)
the operating system for beginning users ..... starter(1)
wall: write to all users ..... wall(1)
ut_delete_user: add/delete users in utmp file ..... utadduser(3X)
mail, rmail: send mail to users or read mail ..... mail(1)
login attempts /usr/adm/loginlog: log of failed/ ..... loginlog(4)
ustat: get filesystem statistics ..... ustat(2)
add/delete users in utmp file ut_add_user, ut_delete_user: ..... utadduser(3X)
utmp file ut_add_user, ut_delete_user: add/delete users in ..... utadduser(3X)
introduction to system standalone utilities ..... intro(8)
get_process_stats: get resource utilization information ..... get_process_stats(2SEQ)
modification times utime: set file access and ..... utime(2)
utmp, wtmp: utmp and wtmp entry formats ..... utmp(4)
ut_delete_user: add/delete users in utmp file ..... utadduser(3X)
endutent, utmpname, getutpid: access utmp file entry ..... getut(3C)
ttyslot: find the slot in the utmp file of the current user ..... ttyslot(3C)
formats utmp, wtmp: utmp and wtmp entry ..... utmp(4)
/pututline, setutent, endutent, utmpname, getutpid: access utmp file/ ..... getut(3C)
and permissions file uucheck: check the uucp directories ..... uucheck(1M)
the uucp system uucico: file transport program for ..... uucico(1M)
clean-up uucleanup: uucp spool directory ..... uucleanup(1M)

uucheck: check the uucp directories and permissions file.....uucheck(1M)
uusched: the scheduler for the uucp file transport program.....uusched(1M)
uucp: networking support files.....uucp(4)
uucleanup: uucp spool directory clean-up.....uucleanup(1M)
uustat: uucp status inquiry and job control.....uustat(1C)
file transport program for the uucp system.....uucico(1M)
UNIX-to-UNIX system/ uucp, uulog, uuname:.....uucp(1C)
speed, and line discipline uugetty: set terminal type, modes,.....uugetty(1M)
uucp, uulog, uuname: UNIX-to-UNIX/.....uucp(1C)
uucp, uulog, uuname: UNIX-to-UNIX system copy.....uucp(1C)
uuto, uupick: public UNIX-to-UNIX/.....uuto(1C)
file transport program uusched: the scheduler for the uucp.....uusched(1M)
control uustat: uucp status inquiry and job.....uustat(1C)
UNIX-to-UNIX system/ uuto, uupick: public.....uuto(1C)
with debugging on Uutry: try to contact remote system.....uutry(1M)
system command execution uux: UNIX-to-UNIX.....uux(1C)
requests uuxqt: execute remote command.....uuxqt(1M)
val: validate SCCS file.....val(1)
val: validate SCCS file.....val(1)
valloc: aligned memory allocator.....valloc(3SEQ)
abs: return integer absolute value.....abs(3C)
u3b5, vax: get processor type truth value.....machid(1)
getenv: return value for environment name.....getenv(3C)
floor, ceiling, remainder, absolute value functions.....floor(3M)
readlink: read value of a symbolic link.....readlink(2SEQ)
putenv: change or add value to environment.....putenv(3C)
current form field to its default values.....reset(1F)
true, false: provide truth values.....true(1)
values: machine-dependent values.....values(5)
values: machine-dependent values.....values(5)
print formatted output of a varargs argument list.....vprintf(3S)
list varargs: handle variable argument.....varargs(5)
varargs: handle variable argument list.....varargs(5)
fpathconf: get configurable pathname variables.....pathconf(2)
sysconf: get configurable system variables.....sysconf(2)
.environ, .pref, .variables: FACE files.....env(4)
unset: set and unset environment variables in core or in files.....set(1F)
edit: text editor (variant of ex for casual users).....edit(1)
proc_ctl: control over various process attributes.....proc_ctl(2SEQ)
/ns32000, pdp11, u3b, u3b2, u3b5, vax: get processor type truth value.....machid(1)
vc: version control.....vc(1)
get option letter from argument vector.....getopt(3C)
readv: read vector from file.....readv(2SEQ)
writev: write vector on a file.....writev(2SEQ)
assert: verify program assertion.....assert(3X)
vc: version control.....vc(1)
get: get a version of an SCCS file.....get(1)
sccsdiff: compare two versions of an SCCS file.....sccsdiff(1)
virtual memory efficient way vfork: spawn new process in a.....vfork(2)
output of a varargs/ vprintf, vfprintf, vsprintf: print formatted.....vprintf(3S)
editor based on ex vi: screen-oriented (visual) display.....vi(1)
more: view a file one full screen at a time.....more(1)

map physical addresses into processs — virtual address space ..... pmap(7)
devtab: — virtual device table ..... devtab(4)
devdestroy: build and destroy — virtual devices ..... devbuild(1M)
vfork: spawn new process in a — virtual memory efficient way ..... vfork(2)
vm_ctl: examine and change — virtual memory tuning parameters ..... vm_ctl(2SEQ)
echo: put string on — virtual output ..... echo(1F)
controlling terminal fvhangup: — virtually hangup the specified ..... fvhangup(2SEQ)
vmtune: tune system-wide — virtual-memory parameters ..... vmtune(1M)
pmap: display or alter parameters of — virtual-to-physical mapping devices ..... pmap(1M)
vi: screen-oriented — visual display editor based on ex ..... vi(1)
memory tuning parameters — vm_ctl: examine and change virtual ..... vm_ctl(2SEQ)
virtual-memory parameters — vmtune: tune system-wide ..... vmtune(1M)
filesystem — volcopy: make literal copy of ..... volcopy(1M)
fs, inode: format of file system — volume ..... fs(4)
formatted output of a varargs/ — vprintf, vfprintf, vsprintf: print ..... vprintf(3S)
the controlling FMLI object — vsig: synchronize a co-process with ..... vsig(1F)
a varargs/ vprintf, vfprintf, — vsprintf: print formatted output of ..... vprintf(3S)
prtvtoc: print the — VTOC of a block device ..... prtvtoc(1M)
prtvtoc: print the — VTOC of a standalone disk device ..... prtvtoc(8)
mkvtoc: populate — VTOC on disks ..... mkvtoc(1M)
vtoc: — VTOC partitioning driver ..... vtoc(7)
— vtoc: VTOC partitioning driver ..... vtoc(7)
s_wait_barrier: initialize barrier, — wait at barrier ..... s_wait_barrier(3PPS)
— wait: await completion of process ..... wait(1)
wait, waitpid: — wait for a child process to stop or terminate ..... wait(2)
sigsuspend: — wait for a signal ..... sigsuspend(2)
process to stop or terminate — wait, waitpid: wait for a child ..... wait(2)
stop or terminate wait, — waitpid: wait for a child process to ..... wait(2)
ftw: — walk a file tree ..... ftw(3C)
— wall: write to all users ..... wall(1)
— wc: word count ..... wc(1)
— wd: SSM SCSI disk controller ..... wd(7)
— what: identify SCCS files ..... what(1)
— whatis: describe what a command is ..... whatis(1)
whodo: — who is doing what ..... whodo(1M)
who: — who is on the system ..... who(1)
— who: who is on the system ..... who(1)
— whodo: who is doing what ..... whodo(1M)
vmtune: tune system- — wide virtual-memory parameters ..... vmtune(1M)
addformopt: add a form/SELECT — window option to a menu ..... addformopt(1M)
wc: — word count ..... wc(1)
fgetc, getw: get character or — word from a stream ..... getc(3S)
fputc, putw: put character or — word on a stream ..... putc(3S)
spellhist: unmatched spelling — words file ..... spellhist(4)
cd: change — working directory ..... cd(1)
chdir: change — working directory ..... chdir(2)
getcwd: get pathname of current — working directory ..... getcwd(3C)
pwd: — working directory name ..... pwd(1)
specific alarms and/or the — working indicator ..... indicator(1F)
interface — wp: SSM parallel line printer ..... wp(7)
lseek: move read/ — write file pointer ..... lseek(2)
tirdwr: Transport Interface read/ — write interface STREAMS module ..... tirdwr(7)

write: write on a file ..... write(2)
putpwent: write password file entry ..... putpwent(3C)
putspent: write shadow password file entry ..... putspent(3X)
wall: write to all users ..... wall(1)
write: write to another user ..... write(1)
writev: write vector on a file ..... writev(2SEQ)
write: write on a file ..... write(2)
write: write to another user ..... write(1)
writev: write vector on a file ..... writev(2SEQ)
open: open for reading or writing ..... open(2)
utmp, wtmp: utmp and wtmp entry formats ..... utmp(4)
utmp, wtmp: utmp and wtmp entry formats ..... utmp(4)
accounting records fwtmp, wtmpfix: manipulate connect ..... fwtmp(1M)
and execute command xargs: construct argument list(s) ..... xargs(1)
multiplexor driver xs: Xylogics 781 terminal ..... xs(7)
tape interface xt: Xylogics 472 MULITBUS magnetic ..... xt(7)
xt: Xylogics 472 MULITBUS magnetic tape interface ..... xt(7)
xs: Xylogics 781 terminal multiplexor driver ..... xs(7)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
yacc: yet another compiler-compiler ..... yacc(1)
compress, uncompress, zcat: compress and expand data ..... compress(1)
zd: dual-channel disk controller ..... zd(7)
for dual-channel disk controller zdformat: standalone disk formatter ..... zdformat(8)
getzdinfo, getzdinfobyname: get zdinfo structure ..... getzdinfo(3SEQ)

NAME

intro – introduction to file formats

DESCRIPTION

This section outlines the formats of various files. The C structure declarations for the file formats are given where applicable.

NAME

a.out – common assembler and link editor output

SYNOPSIS

#include <a.out.h>

DESCRIPTION

The file name **a.out** is the default output file name from the link editor **ld**(1). The link editor makes **a.out** executable if there were no errors in linking. The output file of the assembler **as**(1), also follows the common object file format of the **a.out** file although the default file name is different.

A common object file consists of a file header, a system header (if the file is link editor output), a table of section headers, section data, relocation information, (optional) line numbers, a symbol table, and a string table. The order is given below.

File header.
system header.
Section 1 header.
...
Section n header.
Section 1 data.
...
Section n data.
Section 1 relocation.
...
Section n relocation.
Section 1 line numbers.
...
Section n line numbers.
Symbol table.
String table.

The last three parts of an object file (line numbers, symbol table and string table) may be missing if the program was linked with the **–s** option of **ld**(1) or if they were removed by **strip**(1). Also note that the relocation information is absent after linking unless the **–r** option of **ld**(1) was used. The string table exists only if the symbol table contains symbols with names longer than eight characters.

The sizes of each section (contained in the header, discussed below) are in bytes.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. On Sequent systems the text segment starts at location 0x0.

The **a.out** file produced by **ld**(1) has the magic number 0524 in the first field of the system header. The headers (file header, system header, and section headers) are loaded at the beginning of the text segment and the text immediately follows the headers in the user address space. The first text address equals 0x0 plus the size of the headers, and varies depending upon the

number of section headers in the **a.out** file. In an **a.out** file with four sections (.text, .data, .bss, and .comment), the first text address is at 0xd0 on Sequent systems. The comment section is not loaded in memory and is not part of the process image. The text segment is not writable by the program; if other processes are executing the same **a.out** file, the processes share a single text segment. If the **-Z** option is specified with **ld**, the text starts at hexadecimal 0x1000.

The data segment starts at the next page boundary past the last text address. The first data address is determined by the following: If an **a.out** file were split into 4K blocks, one of the blocks would contain both the end of text and the beginning of data. When the core image is created, that block appears twice; once at the end of text and once at the beginning of data. The duplicated block of text that appears at the beginning of data is never executed; it is duplicated so that the operating system may bring in pieces of the file in multiples of the page size without having to realign the beginning of the data section to a page boundary. Therefore the first data address is the sum of the next segment boundary past the end of text plus the remainder of the last text address divided by 4K. If the last text address is a multiple of 4K, text information is not duplicated.

On Sequent systems the stack begins at location 0x40000000 and grows toward lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the **brk**(2) system call.

For relocatable files, the value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that appears in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, there is a relocation entry for the word, the storage class of the symbol-table entry for the symbol is marked as an "external symbol", and the value and section number of the symbol-table entry are undefined. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol is be added to the word in the file.

File Header

The format of the **filehdr** header is as follows:

```
struct filehdr
{
  unsigned short  f_magic; /* magic number */
  unsigned short  f_nscns; /* number of sections */
  long    f_timdat;    /* time and date stamp */
  long    f_symptr;    /* file ptr to symtab */
  long    f_nsyms;     /* # symtab entries */
  unsigned short  f_opthdr; /* sizeof(opt hdr) */
  unsigned short  f_flags; /* flags */
};
```

System Header

The format of the system header is as follows:

```
typedef struct aouthdr
{
  short magic;          /* magic number */
  short vstamp;         /* version stamp */
```

```
  long  tsize;          /* text size in bytes, padded */
  long  dsize;          /* initialized data (.data) */
  long  bsize;          /* uninitialized data (.bss) */
  long  entry;          /* entry point */
  long  text_start;     /* base of text used for this file */
  long  data_start;     /* base of data used for this file */
} AOUTHDR;
```

Section Header

The format of the section header is as follows:

```
struct scnhdr
{
  char    s_name[SYMNMLEN]; /* section name */
  long    s_paddr;      /* physical address */
  long    s_vaddr;      /* virtual address */
  long    s_size;       /* section size */
  long    s_scnptr;     /* file ptr to raw data */
  long    s_relptr;     /* file ptr to relocation */
  long    s_lnnoptr;    /* file ptr to line numbers */
  unsigned short  s_nreloc; /* # reloc entries */
  unsigned short  s_nlnno; /* # line number entries */
  long    s_flags;      /* flags */
};
```

Relocation

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it is in the following format:

```
struct reloc
{
      long      r_vaddr; /* (virtual) address of reference */
      long      r_symndx;/* index into symbol table */
      ushort    r_type;  /* relocation type */
};
```

The start of the relocation information is **s_relptr** from the section header. If there is no relocation information, **s_relptr** is 0.

Symbol Table

The format of each symbol in the symbol table is as follows:

```
#define  SYMNMLEN 8
#define  FILNMLEN 14
#define  DIMNUM   4

struct syment
{
  union                 /* all ways to get a symbol name */
  {
      char              _n_name[SYMNMLEN]; /* name of symbol */
      struct
      {
          long          _n_zeros;
                        /* == 0L if in string table */
```

```
            long        _n_offset;
                           /* location in string table */
        } _n_n;
        char            *_n_nptr[2];
                           /* allows overlaying */
    } _n;
    long                n_value;
                           /* value of symbol */
    short               n_scnum;
                           /* section number */
    unsigned short      n_type;
                           /* type and derived type */
    char                n_sclass;
                           /* storage class */
    char                n_numaux;
                           /* number of aux entries */
;

#define  n_name  _n._n_name
#define  n_zeros _n._n_n._n_zeros
#define  n_offset          _n._n_n._n_offset
#define  n_nptr  _n._n_nptr[1]
```

Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format is as follows:

```
union auxent {
     struct {
          long  x_tagndx;
          union {
               struct {
                    unsigned shortx_lnno;
                    unsigned shortx_size;
               } x_lnsz;
               long  x_fsize;
          } x_misc;
          union {
               struct {
                    long  x_lnnoptr;
                    long  x_endndx;
               }  x_fcn;
               struct {
                    unsigned shortx_dimen[DIMNUM];
               } x_ary;
          } x_fcnary;
          unsigned short  x_tvndx;
     } x_sym;

     struct {
          char  x_fname[FILNMLEN];
     } x_file;
```

```
        struct {
            long        x_scnlen;
            unsigned short  x_nreloc;
            unsigned short  x_nlinno;
        } x_scn;

        struct {
            long          x_tvfill;
            unsigned short      x_tvlen;
            unsigned short      x_tvran[2];
        } x_tv;
    };
```

Indexes of symbol table entries begin at *zero*. The start of the symbol table is **f_symptr** (from the file header) bytes from the beginning of the file. If the symbol table is stripped, **f_symptr** is 0. The string table (if one exists) begins at `f_symptr + (f_nsyms * SYMESZ)` bytes from the beginning of the file.

SEE ALSO

as(1), cc(1), ld(1), brk(2), filehdr(4), ldfcn(4), linenum(4), reloc(4), scnhdr(4), syms(4)

NAME

acct – per-process accounting file format

SYNOPSIS

#include <sys/acct.h>

DESCRIPTION

Files produced as a result of calling **acct**(2) have records in the form defined by **<sys/acct.h>**, whose contents are:

```
typedef  ushort comp_t;  /* "floating-point" */
             /* 13-bit fraction, 3-bit exponent  */

struct  acct
{
        char    ac_flag;    /* Accounting flag */
        char    ac_stat;    /* Exit status */
        uid_t   ac_uid;     /* Accounting user ID */
        gid_t   ac_gid;     /* Accounting group ID */
        dev_t   ac_tty;     /* control typewriter */
        time_t  ac_btime;   /* Beginning time */
        comp_t  ac_utime;   /* acctng user time in clock ticks */
        comp_t  ac_stime;   /* acctng system time in clock ticks */
        comp_t  ac_etime;   /* acctng elapsed time in clock ticks */
        comp_t  ac_mem;     /* memory usage in clicks */
        comp_t  ac_io;      /* chars trnsfrd by read/write */
        comp_t  ac_rw;      /* number of block reads/writes */
        char    ac_comm[8]; /* command name */
};

extern  struct acct         acctbuf;
extern  struct inode        *acctp;  /* inode of accounting file */

#define AFORK   01          /* has executed fork, but no exec */
#define ASU     02          /* used superuser privileges */
#define ACCTF   0300        . /* record type: 00 = acct */
```

In **ac_flag**, the AFORK flag is turned on by each **fork**(2) and turned off by an **exec**(2). The **ac_comm** field is inherited from the parent process and is reset by any **exec**.

The **ac_mem** field is not maintained by the system and is always reported as 0.

The structure **tacct**, which resides with the source files of the accounting commands, represents the total accounting format used by the various accounting commands:

```
*
 *  total accounting (for acct period), also for day
 */
```

```
struct tacct {
  uid_t          ta_uid;      /* userid */
  char           ta_name[8];  /* login name */
  float          ta_cpu[2];   /* cum. cpu time, p/np (mins) */
  float          ta_kcore[2]; /* cum kcore-minutes, p/np */
  float          ta_con[2];   /* cum. connect time, p/np, mins */
  float          ta_du;       /* cum. disk usage */
  long           ta_pc;       /* count of processes */
  unsigned short ta_sc;       /* count of login sessions */
  unsigned short ta_dc;       /* count of disk samples */
  unsigned short ta_fee;      /* fee for special services */
};
```

SEE ALSO

acct(2), exec(2), fork(2), acct(1M), acctcom(1)

NAME

addusr.def – add user defaults

DESCRIPTION

The Add User Defaults file contains the defaults displayed on the Add User Form.

Each record is in the following format:

name=value

The following default values can be defined:

PASSWD_AGE_MAX
: This record contains the default value for the maximum number of days a password can be in effect.

PASSWD_AGE_MIN
: This record contains the default value for the minimum number of days a password can be in effect.

USERACCT
: This record contains the name of the parent directory where user accounts will be created if a directory is not specified by the System Administrator.

USERGROUP
: This record contains the name of the default group for a new user. This value is displayed in the group field in the **addusr** form.

USERSHELL
: This record contains the name of the default shell or program for a new user. This value is displayed in the shell field in the **addusr** form.

USERMENU
: This record contains the name of the initial menu the user will enter when he logs onto the system. This only applies to users who have the menu system as their initial login program. This value is displayed in the initial menu field in the **addusr** form.

FIELDS

Name

This is the name of the value to be set. See the above description section for the supported values. This is a required field.

Value

This is the default value to be displayed. See the above description section for more information. This is a required field.

FILES

${TFADMFILE}/etc/addusr.def	Add User Defaults

NOTES

To change the contents of this file, use the **addusrdef** command.

NAME

pathalias – alias file for FACE

DESCRIPTION

The **pathalias** files contain lines of the form "alias=path" where *path* is one or more colon (:) separated directories. Whenever a FACE user references a path not beginning with a slash (/), this file is checked. If the first component of the pathname matches the left side of the equals sign, the right side is searched much like the **$PATH** variable in the operating system. This allows users to reference the folder *$HOME/FILECABINET* by typing *filecabinet*.

There is a system-wide pathalias file called **$VMSYS/***pathalias*, and each user can also have local alias file called $HOME/pref/pathalias. Settings in the user alias file override settings in the system-wide file. The system-wide file is shipped with several standard FACE aliases, such as filecabinet, wastebasket, preferences, other_users, etc.

NOTES

Unlike command keywords, partial matching of a path alias is not permitted, however, path aliases are case insensitive. The name of an alias should be alphabetic, and in no case can it contain special characters like "/", " or "=". There is no limit on the number of aliases allowed. Alias files are read once, at login, and are held in core until logout. Thus, if an alias file is modified during a session, the change does not take effect until the next session.

FILES

$HOME/pref/pathalias
$VMSYS/pathalias

SEE ALSO

env(4)

NAME

backup.codes – backup cycle code definitions

DESCRIPTION

The backup cycle codes file lists the types of backup cycles available for scheduled backups.

FIELDS

Backup Cycle
Type of backup cycle. This field is the key to the record. This is a required field.

Backup Cycle Description
This is the long description of the cycle displayed in SELECT windows. This is a required field.

Backup Cycle Code
This code describes the length of the backup cycle to the menu system, which supports the following codes:

W Weekly Backup Cycle
M Monthly Backup Cycle
Y Yearly Backup Cycle

This is a required field.

Retention Multiplier
Not currently used. This is an optional field.

Is epoch date based on fiscal
Not currently used. This is an optional field.

Epoch Date
Start date of the period. This is a required field.

FILES

${TFADMFILE}/etc/backup.codes	Backup Cycle Code Definitions
${TFADMFILE}/etc/backup.codes1	Backup Subcodes Description File

NAME

backup.codes1 – backup subcode definitions

DESCRIPTION

The backup subcodes file contains the components of each backup cycle defined on the system.

FIELDS

Backup Cycle
Type of backup cycle. One or more records in this file can be related to the corresponding cycle definition in the **backup.codes** file. This is a required field.

Backup Cycle Description
This a description of the cycle displayed in SELECT windows. This is a required field.

Backup Cycle Subcode
This is a code that describes the subinterval of the backup cycle period. For a weekly cycle, the subcodes are abbreviations for the days of the week. For a monthly cycle, the subcodes are the days of the month (that is, 1 through 31). For a yearly cycle, the subcodes are the months of the year. This is a required field.

Subcode Description
This is a description of the subcode displayed to the user. The subcode is used internally only. This is a required field.

Day within cycle
This is the day when the subinterval of the cycle occurs. This is a required field.

FILES

${TFADMFILE}/etc/backup.codes1	Backup Subcode Definitions
${TFADMFILE}/etc/backup.codes	Backup Cycle Code Definitions

NAME

backup.dev – backup devices

DESCRIPTION

The commands on the Backup and Restore Menu use this file. The file is maintained by the forms on the Backup Device Information Menu, a submenu of the Backup and Restore Menu.

FIELDS

Raw Device
This is the name of the raw device for the backup device. This is a required field.

Block Device
This is the name of the raw device for the backup device. This is a required field.

Mountable Device?
This is a boolean code that tells you if the device can be mounted by the backup system. The menu system supports the following codes:

Y Yes
N No

This is a required field.

Number of Blocks
Capacity of the media in disk blocks. Block size is 512 bytes. The value in this field controls the size of the file system that can be built on the device. This is an optional field.

Number of Inodes
This is the number of inodes that are allocated when a file system is built. This is an optional field.

Description
Description of the backup device. The value in this field is displayed in the SELECT windows that display backup devices. The text should be easy to understand. This is a required field.

Input Preprocessor Program
This program preprocesses the input and pipes it to the **cpio –i** program. The **cpio –i** then reads the information. This is an optional field.

Output Postprocessor Program
This program postprocesses the output of the **cpio -o** program and writes directly to the output device. This is an optional field.

Tape Rewinder Program
This option is for sequential devices that support a rewinding function (for example, tape drives). It is not applicable to random access devices such as hard disks. This is a optional field.

Media Format Command
This option is for devices that must be formatted before use. Enter the command and the options required to format the device. This is a optional field.

Tape Retension Program
This option is for sequential devices that support a retensioning function (for

example, tape drives). This option does not apply to random access devices such as hard disks. This is a optional field.

Media Erase Command
This option is for devices that support an erase function. This is a optional field.

FILES

${TFADMFILE}/etc/backup.dev	Backup Devices

NAME

backup.info – backup set definitions

DESCRIPTION

The Backup Set Definitions file lists the backup operations defined by users in the Scheduled Backup Management menu.

FIELDS

Backup Set Name
This name identifies the backup set. This is a required field.

Backup Set Description
This field is displayed in the SELECT windows and helps the user identify the backup set. This is a required field.

Directories to be backed up
This code describes the length of the backup cycle. The system supports the following codes:

W Weekly Backup Cycle
M Monthly Backup Cycle
Y Yearly Backup Cycle

This is a required field.

Retention Multiplier
Not currently used. This is an optional field.

Is epoch date based on fiscal
Not currently used. This is an optional field.

Epoch Date
Start date of the period. This is a required field.

FILES

${TFADMFILE}/etc/backup.info	Backup Set Definitions
${TFADMFILE}/etc/backup.codes1	Backup Subcodes Description File

NAME

cftime – language specific strings

DESCRIPTION

The programmer can create one printable file per language. These files must be kept in a special directory, */lib/cftime*. If this directory does not exist, the programmer should create it. The contents of these files are:

- abbreviated month names (in order)
- month names (in order)
- abbreviated weekday names (in order)
- weekday names (in order)
- default strings that specify formats for local time (%x) and local date (%X).
- default format for cftime, if the argument for cftime is zero or null.
- AM (ante meridian) string
- PM (post meridian) string

Each string is on a line by itself. All white space is significant. The order of the strings in the above list is the same order in which the strings appear in the file shown below.

EXAMPLE

```
/lib/cftime/usa_english

Jan
Feb
...
January
February
...
Sun
Mon
...
Sunday
Monday
...
%H:%M:%S
%m/%d/%y
%a %b %d %T %Z %Y
AM
PM
```

FILES

/lib/cftime – directory that contains the language specific printable files (create it if it does not exist)

SEE ALSO

ctime(3C)

NAME

checklist – list of filesystems processed by fsck and ncheck

DESCRIPTION

checklist resides in directory */etc* and contains a list of special filenames. Each special filename is contained on a separate line and corresponds to a filesystem. Each filesystem is then automatically processed by the **fsck**(1M) command if no filesystem arguments are present.

FILES

/etc/checklist

SEE ALSO

fsck(1M), ncheck(1M)

NAME

constab – constructed devices table

DESCRIPTION

constab resides in directory */etc* and contains a table of virtual devices which have been built by the **devbuild**(1M) command. The **devdestroy**(1M) command removes entries from this file.

The file contains a line of information for each constructed device, and is structurally similar to the contents of */etc/devtab* as described in **devtab**(4), with the exception that only the first two columns are represented.

The entries from the **constab** file are accessed using the routines in **read_constab**(3SEQ) or printed via **devbuild**(1M).

FILES

/etc/constab

SEE ALSO

devbuild(1M), devdestroy(1M), read_constab(3SEQ), devtab(4)

NAME

core – format of core image file

DESCRIPTION

The operating system writes out a memory image of a terminated process when any of various errors occur (see **signal**(2) for a list of errors). The most common errors are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called **core** and is written in the process's working directory, provided access it allowed. A process with an effective user ID different from the real user ID, or an effective group ID different from the real group ID, will not produce a core image.

The first section of a core image is a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. The size of this section depends on the parameter UPAGES, which is defined in *<sys/mc_param.h>*. The remainder represents the actual contents of the user's core area when the core image was written. If the text segment is read-only and shared, or separated from data space, it is not dumped.

The format of the information in the first section is described by the *user* structure of the system, defined in *<sys/user.h>*. The location of the registers are outlined in *<sys/reg.h>*.

SEE ALSO

pdbx(1), signal(2), sigaction(2)

NAME

cpio – format of cpio archive

DESCRIPTION

The header structure, when the **–c** option of **cpio**(1) is not specified, is as follows:

```
struct {
            short   h_magic,
                    h_dev;
            ushort  h_ino,
                    h_mode,
                    h_uid,
                    h_gid;
            short   h_nlink,
                    h_rdev,
                    h_mtime[2],
                    h_namesize,
                    h_filesize[2];
            char    h_name[h_namesize rounded to word];
} Hdr;
```

If **–c** is specified, the header information is described by the following:

```
sscanf(Chdr,"%6o%6o%6o%6o%6o%6o%6o%11o%6o%11o%s",
       &Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
       &Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
       &Longtime, &Hdr.h_namesize,&Longfile,Hdr.h_name);
```

Longtime and *Longfile* are equivalent to *Hdr.h_mtime* and *Hdr.h_filesize*, respectively. The contents of each file are recorded in an element of the array of varying length structures, *archive*, together with other items describing the file. Every instance of *h_magic* contains the constant 070707 (octal). The items *h_dev* through *h_mtime* have meanings explained in **stat**(2). The length of the null-terminated pathname *h_name*, including the null byte, is given by *h_namesize*.

The last record of the *archive* always contains the name TRAILER. Special files, directories, and the trailer are recorded with *h_filesize* equal to zero.

SEE ALSO

stat(2), cpio(1), find(1)

NAME

cronlog – cron history file

DESCRIPTION

A history of all actions taken by **/etc/cron** is recorded in the **/usr/lib/cron/log** file. The **/usr/lib/cron/log** file should be periodically truncated to keep the size of the file within a reasonable limit. Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **/usr/lib/cron/log** file. To use one of these functions to limit the size of a log file, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/ rc2.d, rc3.d ...** file, as applicable. The following command line limits the size of the log file to the last 100 lines in the file:

```
tail -100 /usr/lib/cron/log > /tmp/log; mv /tmp/log /usr/lib/cron/log
```

EXAMPLES

The following is an example of a typical */usr/lib/cron/log* file.

```
! *** cron started ***   pid = 237 Sun Aug 19 14:06:45 1989
>  CMD: /usr/lib/uucp/uudemon.hour > /dev/null
>  root 251 c Sun Aug 19 14:11:00 1984
<  root 251 c Sun Aug 19 14:11:01 1984
>  CMD: /usr/lib/uucp/uudemon.poll > /dev/null
>  root 370 c Sun Aug 19 14:30:00 1984
<  root 370 c Sun Aug 19 14:30:03 1984
>  CMD: /usr/lib/uucp/uudemon.hour > /dev/null
>  root 417 c Sun Aug 19 14:41:01 1984
<  root 417 c Sun Aug 19 14:41:02 1984
>  CMD: /usr/lib/uucp/uudemon.poll > /dev/null
>  root 452 c Sun Aug 19 15:01:00 1984
<  root 452 c Sun Aug 19 15:01:04 1984
>  CMD: /usr/lib/uucp/uudemon.hour > /dev/null
>  root 460 c Sun Aug 19 15:11:00 1984
<  root 460 c Sun Aug 19 15:11:00 1984
>  CMD: /usr/lib/uucp/uudemon.poll > /dev/null
>  root 541 c Sun Aug 19 15:30:00 1984
<  root 541 c Sun Aug 19 15:30:07 1984
```

NAME

crontabs – clock daemon file

DESCRIPTION

The **/usr/spool/cron/crontabs** directory contains **crontab** files for **adm**, **root**, and **sys** logins. Providing their lognames are in the */usr/lib/cron/cron.allow* file, users can establish their own **crontabs** file using the **crontab** command. If the *cron.allow* file does not exist, the */usr/lib/cron/cron.deny* file is checked to determine if the user is denied the use of the **crontab** command.

As **root**, you can either use the **crontab**(1) command or edit the appropriate file under */usr/spool/cron/crontabs* to make the desired entries. Revisions to the file take effect at the next reboot.

The line entry format of a */usr/spool/cron/crontabs/logname* file is as follows:

minute hour day month day-of-week command

The various fields of a *crontabs/logname* line entry are as follows:

minute The minutes field is a one- or two-digit number in the range 0 through 59.

hour The hour field is a one- or two-digit number in the range 0 through 24.

day The day field is the numerical day of the month in the range 1 through 31.

month The month field is the numerical month of the year in the range 1 through 12.

day-of-week The day-of-week field is the numerical day of the week where Sunday is 0, Monday is 1, . . . and Saturday is 6.

command The command field is the program or command that is executed at the time specified by the first five fields.

The following syntax applies to the first five fields:

- Two numbers separated by a minus indicates an inclusive range of numbers between the two specified numbers.
- A list of numbers separated by commas specifies all of the numbers listed.
- An asterisk specifies all legal values.

In the command field (sixth field), a percent sign (%) is translated to a newline character. Only the first line of a command field (character string up to the percent sign) is executed by the shell. Any other lines are made available to the command as standard input. The example that follows shows a typical */usr/spool/cron/crontabs/logname* file. The data shown are the *root* file. The file entries support the **calendar** reminder service and basic networking. Remember, you can use the **cron** function to decrease the number of data terminal driven system administration tasks: include recurring and habitual

tasks in your crontab file.

EXAMPLES

```
0 1 * * * /usr/bin/calendar -
41,11 * * * * /usr/lib/uucp/uudemon.hour > /dev/null
45 23 * * * ulimit 5000; /bin/su uucp -c "/usr/uudemon.cleanup"
  > /dev/null 2>&1
1,30 * * * * /usr/lib/uucp/uudemon.poll > /dev/null
```

NAME

devtab – virtual device table

DESCRIPTION

The */etc/devtab* file contains information about virtual device bindings for use by **devbuild**(1M). Each entry in */etc/devtab* has the following format.

Column 1	Virtual device type
Column 2	Root of virtual device tree
Column 3	Parameters to device builder

Virtual device type is one of a set of supported virtual devices. Each virtual device type is supported by an executable in */etc/devbuild.d*, so the list of supported virtual devices can be listed by examining this directory. The system is distributed with two virtual device types: **vtoc**(7), a partitioning driver based on the "Volume Table of Contents" (VTOC) present on the disk, and **cmpt**(7), a partitioning driver which supplies compatibility partitions for previous versions of the system. Refer to each virtual driver's manual page for the format of their respective *devtab* entries.

FILES

/etc/devtab
/etc/devbuild.d

SEE ALSO

devbuild (1M), devdestroy (1M), constab (4), vtoc (7), cmpt (7)

NAME

dir – format of directories

SYNOPSIS

#include <sys/types.h>
#include <sys/ufsdir.h>

DESCRIPTION

A directory behaves exactly like an ordinary file, except that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry (see **fs**(4)). The structure of a directory entry as given in the include file is:

```
/*
 * A directory consists of some number of blocks of
 * DIRBLKSIZ bytes, where DIRBLKSIZ is chosen such
 * that it can be transferred * to disk in a single
 * atomic operation (e.g. 512 bytes on most machines).
 * Each DIRBLKSIZ byte block contains some number of
 * directory entry structures, which are of variable
 * length.  Each directory entry has a struct direct
 * at the front of it, containing its inode number,
 * the length of the entry, and the length of the
 * name contained in * the entry.  These are followed
 * by the name padded to a 4 byte boundary with null
 * bytes.  All names are guaranteed null terminated.
 * The maximum length of a name in a directory is
 * UFS_MAXNAMLEN.
 *
 * The macro DIRLEN(dp) gives the amount of space
 * required to represent a directory entry.  Free
 * space in a directory is represented by entries
 * which have dp->d_reclen > DIRLEN(dp).  All
 * DIRBLKSIZ bytes in a directory block are claimed
 * by the directory entries.  This usually results
 * in the last entry in a directory having a large
 * dp->d_reclen.  When entries are deleted from a
 * directory, the * space is returned to the previous
 * entry in the same directory block by increasing its
 * dp->d_reclen.  If the first entry of a directory
 * block is free, then its dp->d_ino is set to 0.
 * Entries other than the first in a directory do not
 * normally have dp->d_ino set to 0.
 */
#define DIRBLKSIZ DEV_BSIZE

#define UFS_MAXNAMLEN 255

/*
 * The DIRLEN macro gives the minimum record length
 * which will hold the directory entry.  This requires
 * the amount of space in struct direct without the
 * d_name field, plus enough space for the name with
```

```
 * a terminating null byte (dp->d_namlen+1), rounded
 * up to a 4 byte boundary.
 */
#undef DIRLEN
#define DIRLEN(dp) \
    ((sizeof (struct direct) - (UFS_MAXNAMLEN+1)) +
(((dp)->d_namlen+1 + 3) &~ 3))

struct direct {
       ulong    d_ino;
       short    d_reclen;
       short    d_namlen;
       char     d_name[UFS_MAXNAMLEN + 1];
       /* typically shorter */
};
```

By convention, the first two entries in each directory are for '.' and '..'. The first is an entry for the directory itself. The second is for the parent directory. The meaning of '..' is modified for the root directory of the master file system (/), where '..' has the same meaning as '.'.

Applications which read directories to get filenames from it should use the **opendir**(3X), and **readdir(3X)** library calls.

SEE ALSO

fs(4), dirent(4), directory(3x)

NAME

dirent – file system independent directory entry

SYNOPSIS

#include <sys/dirent.h>
#include <sys/types.h>

DESCRIPTION

Different file system types may have different directory entries. The **dirent** structure defines a file system independent directory entry, which contains information common to directory entries in different file system types. A set of these structures is returned by the **getdents**(2) system call.

The **dirent** structure is defined below.

```
structdirent {
                    long                d_ino;
                    off_t               d_off;
                    unsigned short      d_reclen;
                    char                d_name[1];
            };
```

The **d_ino** is a number which is unique for each file in the file system. The field **d_off** is the offset of that directory entry in the actual file system directory. The field **d_name** is the beginning of the character array giving the name of the directory entry. This name is null terminated and may have at most **MAXNAMLEN** characters. This results in file system independent directory entries being variable length entities. The value of **d_reclen** is the record length of this entry. This length is defined to be the number of bytes between the current entry and the next one, so that it always results in the next entry being on a long boundary.

FILES

/usr/include/sys/dirent.h

SEE ALSO

getdents(2)

NAME

diskinfo – disk device characteristic database

SYNOPSIS

/etc/diskinfo/*

DESCRIPTION

diskinfo is a database which describes characteristics of disk devices; that is, those devices which need to be formatted and/or those devices on which a VTOC will be placed. The directory */etc/diskinfo* is populated with text files that describe different classes of disk characteristics.

Each type of disk is assigned a symbolic name which is usually the model number of the particular disk – for example, "m2333k" for the Fujitsu "Swallow 3." Each file in */etc/diskinfo* has a prefix which is the name of a disk type. The suffix of the filename defines the class of disk information present on this particular disk model. For example, *m2333k.geom* will contain the geometry information for all model M2333K disks.

Lines in **diskinfo** files are patterned after the **terminfo**(4) database. Below is a general description of lines in these files.

diskinfo files consist of usually one device description entry. Each entry consists of a header (beginning in column 1) and one or more lines that list the features for that particular device. Each line in a **diskinfo** file, except the header, must be indented with one or more white spaces (either spaces or tabs).

Entries consist of a number of colon-separated fields. The following example shows the format of a **diskinfo** source file:

alias1 | alias2 | ... | aliasN | *longname*:
name1#value1:name2#value2: ...

The first line, commonly referred to as the header line, must begin in column one and must contain one or more aliases, separated by vertical bars. The last field in the header line may be the long name of the device and it may contain any string. Alias names must be unique within files of a given suffix in the **diskinfo** database and they must conform to operating system file naming conventions; for example they cannot contain white space or slashes.

GEOMETRY INFORMATION

Files in */etc/diskinfo* with the suffix *.geom* contain disk geometry information, such as the size of tracks and cylinders, and the size of the disks themselves. This information is usually common to all disks, and is used by the system to optimize access to data by grouping related disk blocks relatively close together. The information is also used by system maintenance utilities (such as formatters) so that they can agree on reserved disk areas.

The following list indicates the normal values stored for each disk entry.

Name	Type	Description
cy	num	Number of sectors per cylinder
dc	num	Capacity of disk in sectors
ns	num	Number of sectors per track
nt	num	Number of tracks per cylinder
nc	num	Total number of cylinders on the disk
rm	num	Number of revolutions per minute
se	num	Sector size in bytes (512 by default)
xc	num	Minimum disk capacity in sectors

Note that even though the number of sectors per track and tracks per cylinder are given, the number of sectors per cylinder is not necessarily equal to the **ns** and **nt** values multiplied together. Also, the capacity of the whole disk may not be derivable from any of the other parameters. This is due to the construction of disks which have a variable number of sectors per track depending on the cylinder number, and disks which have extra sectors on each cylinder.

The **xc** parameter describes the guaranteed minimum number of sectors on the disk. This is needed for disk devices which vary their reported size based on the number of bad blocks which have been revectored on the disk. The **xc** parameter is used by programs which need to varify the disk type by seeking to the last known sector of the disk. The **xc** parameter is normally not present for most disk types.

The information in the *.geom* files can be retrieved in a program using the **getgeombyname**(3SEQ) interface.

SCSI INFORMATION

Files in */etc/diskinfo* with the suffix *.scsi* contain information specific to CCS (common command set) SCSI disks which is used by the online format program. This data should correspond to fields in the *drive_table* entries in */etc/conf/stand/conf_scsi.c*.

The following list defines the two character keys used to identify each field within the file. Each of these fields is required by the online formatter to be in the file.

Name	Type	Description
vn	string	Vendor (this must be padded to 8 chars)
pr	string	Product number (this must be padded to 16 chars)
iq	num	Inquiry command format
re	num	Size in bytes of REASSIGN_BLOCK data
fm	num	Format code for the FORMAT command

Note that the vendor and product number must match the strings returned by the SCSI INQUIRY command.

The information in the *.scsi* files can be retrieved in a program using the **getscsimatch**(3SEQ) interface, which requires the vendor and products strings to be passed in as arguments, and then finds the *.scsi* file which matches.

ZD INFORMATION

Files in */etc/diskinfo* with the suffix *.zd* contain information specific to ZDC disks which is used by the online format program. This data should

correspond to fields in the *zdinfo* table entries in */etc/conf/stand/conf_zd.c* and should not be changed, since it used to construct channel configuration information used by the ZDC controller board.

The following list defines the two character keys used to identify each field within the file. Each of these fields is required to be in the file. Compare with the *zdcdd* structure defined in *<sys/zdc.h>*.

Name	Type	Description
mg	num	magic number for sanity
ec	num	number of bytes of ECC
sp	num	spares per track
se	num	sectors per track (formatted)
tr	num	tracks per cylinder
cy	num	number of cylinders
dt	num	soft drive type (type it was formatted as).
xf	num	drive transfer rate in MHz
ru	num	number of bytes in runt sector
ch	num	channel delay
hs	num	head switch delay
hr	num	header postamble byte cnt (read)
hf	num	header postamble byte cnt (fmt)
ck	num	format skew between cylinders
tk	num	format skew between tracks
hb	num	header byte cnt (inc head scatter)
sb	num	sector byte cnt (inc header & gap)
si	num	bytes at sector start where defects can be ignored
ei	num	bytes at sector end where defects can be ignored
ds	num	RO status register
de	num	RO error register
p0	num	polynomial preset byte 0
p1	num	polynomial preset byte 1
p2	num	polynomial preset byte 2
p3	num	polynomial preset byte 3
p4	num	polynomial preset byte 4
p5	num	polynomial preset byte 5
t0	num	polynomial tap byte 0
t1	num	polynomial tap byte 1
t2	num	polynomial tap byte 2
t3	num	polynomial tap byte 3
t4	num	polynomial tap byte 4
t5	num	polynomial tap byte 5
er	num	ECC/CRC control
hc	num	header byte count
dc	num	drive command register
oc	num	operation command register
sc	num	sector count
ns	num	number of sector operations

h0	num	header byte 0 pattern
h1	num	header byte 1 pattern
h2	num	header byte 2 pattern
h3	num	header byte 3 pattern
h4	num	header byte 4 pattern
h5	num	header byte 5 pattern
rd	num	remote data byte count
dm	num	DMA address
do	num	data postamble byte count
rh	num	header preamble byte count
s1	num	header synch #1 byte count
s2	num	header synch #2 byte count
c0	num	header byte 0 control
c1	num	header byte 1 control
c2	num	header byte 2 control
c3	num	header byte 3 control
c4	num	header byte 4 control
c5	num	header byte 5 control
xd	num	external data ECC byte count
xh	num	external header ECC byte count
po	num	header postamble byte count (write)
pr	num	data preamble byte count (write)
d1	num	data synch #1 byte count
d2	num	data synch #2 byte count
op	num	data postamble pattern
or	num	header preamble pattern
a1	num	header synch #1 pattern
a2	num	header synch #2 pattern
gp	num	gap byte count
df	num	disk format register
lt	num	local transfer register
rt	num	remote transfer register
et	num	sector byte count
gt	num	gap pattern
fd	num	data format pattern
ah	num	header postamble pattern
ad	num	data preamble pattern
r1	num	data synch #1 pattern
r2	num	data synch #2 pattern

The information in the *.zd* files can be retrieved in a program using the **getzdinfobyname**(3SEQ) interface.

FILES

/etc/diskinfo/* /etc/conf/stand/conf_scsi.c /etc/conf/stand/conf_zd.c

SEE ALSO

getgeombyname (3SEQ), getscsimatch (3SEQ), getzdinfobyname (3SEQ), mkvtoc (1M), format (1M)

NAME

dump, dumpdates – incremental dump format

SYNOPSIS

#include <sys/types.h>
#include <sys/ufsinode.h>
#include <dumprestore.h>

DESCRIPTION

Tapes used by **dump** and **restore**(1M) contain the following:

a header record
two groups of bit map records
a group of records describing directories
a group of records describing files

The format of the header record and of the first record of each description as given in the include file *<dumprestore.h>* is:

```
#define NTREC       10
#define HIGHDENSITYTREC 32
#define MLEN        16
#define MSIZ        4096

#define TS_TAPE     1
#define TS_INODE    2
#define TS_BITS     3
#define TS_ADDR     4
#define TS_END      5
#define TS_CLRI     6
#define MAGIC       (int) 60012
#define CHECKSUM    (int) 84446

struct spcl {
        int             c_type;
        time_t          c_date;
        time_t          c_ddate;
        int             c_volume;
        daddr_t                 c_tapea;
        ino_t           c_inumber;
        int             c_magic;
        int             c_checksum;
        struct          dinode          c_dinode;
        int             c_count;
        char            c_addr[BSIZE];
        char            c_label[LBLSIZE];
        long            c_level;
        char            c_filesys[NAMELEN];
        char            c_dev[NAMELEN];
        char            c_host[NAMELEN];
} spcl;

struct idates {
        char            id_name[16];
```

```
        char        id_incno;
        time_t      id_ddate;
};

#define     DUMPOUTFMT  "%-16s %c %s"
                        /* for printf */
                        /* name, incno, ctime(date) */
#define     DUMPINFMT   "%16s %c %[^\n]\n"
                        /* inverse for scanf */
```

NTREC is the number of 1024 byte records in a physical tape block, unless a 6250 BPI density or higher tape is used, in which case the number of 1024 byte records in a physical tape block is HIGHDENSITYTREC. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the *c_type* field to indicate what sort of header this is. The types and their meanings are as follows:

TS_TAPE	Tape volume label
TS_INODE	A file or directory follows. The *c_dinode* field is a copy of the disk inode and contains bits telling what sort of file this is.
TS_BITS	A bit map follows. This bit map has a one bit for each inode that was dumped.
TS_ADDR	A subrecord of a file description. See *c_addr* below.
TS_END	End of tape record.
TS_CLRI	A bit map follows. This bit map contains a zero bit for all inodes that were empty on the file system when dumped.
MAGIC	All header records have this number in *c_magic*.
CHECKSUM	Header records checksum to this value.

The fields of the header structure are as follows:

c_type	The type of the header.
c_date	The date the dump was taken.
c_ddate	The date the file system was dumped from.
c_volume	The current volume number of the dump.
c_tapea	The current number of this (1024-byte) record.
c_inumber	The number of the inode being dumped if this is of type TS_INODE.
c_magic	This contains the value MAGIC above, truncated as needed.
c_checksum	This contains whatever value is needed to make the record sum to CHECKSUM.
c_dinode	This is a copy of the inode as it appears on the file system; see *fs*(5).
c_count	The count of characters in *c_addr*.
c_addr	An array of characters describing the blocks of the dumped file. A character is zero if the block associated with that character was not present on the file system, otherwise the character is non-zero. If the block was not present on the file system, no block was dumped; the block is restored as a hole in the file. If

there is not sufficient space in this record to describe all of the blocks in a file, TS_ADDR records are scattered through the file, each one picking up where the last left off.

c_label Contains the dump level, not currently used.

c_level The increment level of this dump.

c_filesys The name of the filesystem (mount point) which was dumped on this tape.

c_dev The name of the disk device which contained the file system which was dumped on this tape.

c_host The name of the host on which the dump took place.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a TS_END record and then the tapemark.

The structure *idates* describes an entry in the file */etc/dumpdates* where dump history is kept. The fields of the structure are:

id_name The dumped filesystem is '/dev/*id_nam*'.

id_incno The level number of the dump tape; see *dump*(8).

id_ddate The date of the incremental dump in system format see *types*(5).

FILES

/etc/dumpdates

SEE ALSO

dump(1M), restore(1M), types(5)

NAME

.environ, .pref, .variables – FACE files

DESCRIPTION

The **.environ** file contains system-wide FACE environment variables. The **.pref** file contains the defaults for WASTEBASKET and FILECABINET. The **.variables** file contains variables to set the editor and the shell prompt. These files are used for a variety of FACE operations.

The **.environ** and **.variables** files are located in the user's *$HOME/pref* directory. The **.pref** files are found under *$HOME/FILECABINET*, *$HOME/WASTEBASKET*, and any directory where preferences were set with the **organize** command. Names and descriptions for each variable are presented below. Variables are listed one per line and are of the form "variable=value".

Variables found in **.environ** are as follows:

LOGINWIN1 ... LOGINWIN4
Windows that are opened when FACE is initialized

SORTMODE Sort mode for file folder listings. Values include the following hexadecimal digits:

1	sort alphabetically by name
2	files most recently modified first
800	sort alphabetically by object type

The values above may be listed in reverse order by "ORing" the following value:

1000 list objects in reverse order

For example, a value of 1002 produces a folder listing with files LEAST recently modified displayed first. A value of 1001 produces a reverse alphabetical by name listing of the folder.

DISPLAYMODE Display mode for file folders. Values include the following hexadecimal digits:

0	file names only
4	file names and brief description
8	file names, description, plus additional information

WASTEPROMPT Prompt before emptying wastebasket (yes/no)?

WASTEDAYS # days before emptying wastebasket

PRINCMD1 ... PRINCMD3
Print command defined to print files

UMASK Holds default permissions that files will be created with.

Variables found in **.pref** are **SORTMODE** and **DISPMODE**, which can have the same values as the **SORTMODE** and **DISPLAYMODE** variables described in **.environ** above.

Variables found in **.variables** are **EDITOR** indicating the default editor, and **PS1** indicating the shell prompt.

FILES

$HOME/pref/.environ
$HOME/pref/.variables
$HOME/FILECABINET/.pref
$HOME/WASTEBASKET/.pref

NAME

errorlog – internal error log file

DESCRIPTION

The error log file contains all internal menu system errors and failed system calls that occur during user activity. The System Admininstrator can inspect this log periodically to review the types of errors, if any, that are occurring. This log is also reviewed when the menu program (menuprom) will not start up or is not working properly.

A single error may result in more than one entry in the error log file. Because the system checks each subroutine's return code, an error may cascade through several levels of subroutine calls before the initial subroutine that was called is notified of the error.

FIELDS

Date
The date the internal error occurred (format: MM/DD/YY). This is a required field.

Time
The time the internal error occurred (format: HH.MM.SS., based on a 24 hour clock). This is a required field.

Message Number
Number assigned to the error. This is a required field.

Executable Program
Name of the executable binary that detected the error. This is a required field.

Subroutine Name
Name of the subroutine that detected the error. This is a required field.

Message Text
The Message Text is extracted from the Error Message file based on the message number. It describes the error that occurred. It may also be displayed on the screen when the error occurs. This is a required field.

UNIX System Error Text
Some system errors fail with a diagnostic number. The diagnostic number is equivalent to the error number for the UNIX **perror()** system call. This is a optional field.

FILES

${TFADMFILE}/misc/errorlog	Interal Error Log File
${TFADMFILE}/misc/errormsg	TaskForce Error Message File

NOTES

Alteration of this file is not recommended. When a system malfunction is detected, inspect this file to uncover the error.

NAME

errormsg – error message text file

DESCRIPTION

The error message file contains the text of all error messages that are the result of internal menu system errors. All messages are referenced by number in the menu system source code. When an internal failure is detected, the text for the error message is retrieved by the **tferror()** routine in the menu system. The error message is then displayed on the screen or written in the menu system's Internal Error Log File.

FIELDS

Message Number

When an error occurs, a message number is passed to the **tferror()** routine. The system searches this file for a record with the same number. This is a required field.

Message Text

The text returned to the **tferror()** routine. It may be displayed on the screen or written to the internal menu system Error Log. This is a required field.

FILES

${TFADMFILE}/etc/errormsg	Error Message Text File
${TFADMFILE}/misc/errorlog	TaskForce Internal Log File

SEE ALSO

menu(1M), edc(1M)

NOTES

The alteration of this file is not recommended.

NAME

filehdr – file header for common object files

SYNOPSIS

#include <filehdr.h>

DESCRIPTION

Every common object file begins with a 20-byte header. The following C **struct** declaration is used:

```
struct filehdr
{
  unsigned short  f_magic ;     /* magic number */
  unsigned short  f_nscns ;     /* number of sections */
  long    f_timdat ;            /* time & date stamp */
  long    f_symptr ;            /* file ptr to symtab */
  long    f_nsyms ;             /* # symtab entries */
  unsigned short  f_opthdr ;    /* sizeof(opt hdr) */
  unsigned short  f_flags ;     /* flags */
} ;
```

f_symptr is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in **fseek**(3S) to position an I/O stream to the symbol table. The optional header is 28-bytes. The valid magic numbers are given below:

```
#define _ 0524   /* Sequent Symmetry computers */
```

The value in **f_timdat** is obtained from the **time**(2) system call. Flag bits currently defined are:

```
#define _ 0000001  /* relocation entries stripped */
#define _ 0000002  /* file is executable */
#define _ 0000004  /* line numbers stripped */
#define _ 0000010  /* local symbols stripped */
#define _ 0000020  /* minimal object file */
#define _ 0000040  /* update file, ogen produced */
#define _ 0000100  /* file is "pre-swabbed" */
#define _ 0000200  /* 16-bit DEC host */
#define _ 0000400  /* 32-bit DEC host */
#define _ 0001000  /* non-DEC host */
#define _ 0002000  /* "patch" list in opt hdr */
#define _ 0160000  /* WE32000 family ID field */
#define _ 0020000  /* file contains WE 32100 code */
#define _ 0040000  /* file reqs MAU to execute */
#define _ 0010000  /* this object file contains restore
                      work around [3B5/3B2 only] */
```

SEE ALSO

time(2), fseek(3S), a.out(4)

NAME

fs, inode – format of file system volume

SYNOPSIS

#include <sys/types.h>
#include <sys/ufsfilsys.h>
#include <sys/ufsinode.h>

DESCRIPTION

Every file system storage volume (disk, nine-track tape, for instance) has a common format for certain vital information. Every such volume is divided into a certain number of blocks. The block size is a parameter of the file system. Sectors beginning at BBLOCK and coutinuing for BBSIZE are used to contain primary and secondary bootstrapping programs.

The actual file system begins at sector SBLOCK with the *super block* that is of size SBSIZE. The layout of the super block as defined by the include file *<sys/ufsfilsys.h>* is:

```
#define     FS_MAGIC    0x011954
structfs {
  struct  fs *fs_link;  /* linked list of file systems */
  struct  fs *fs_rlink; /* used for incore super blocks */
  daddr_t fs_sblkno;    /* addr of superblock in filesys */
  daddr_t fs_cblkno;    /* offset of cyl-block in filesys */
  daddr_t fs_iblkno;    /* offset of inode-blocks in filesys */
  daddr_t fs_dblkno;    /* offset of first data after cg */
  long    fs_cgoffset;  /* cylinder group offset in cylinder */
  long    fs_cgmask;    /* used to calc mod fs_ntrak */
  time_t  fs_time;      /* last time written */
  long    fs_size;      /* number of blocks in fs */
  long    fs_dsize;     /* number of data blocks in fs */
  long    fs_ncg;       /* number of cylinder groups */
  long    fs_bsize;     /* size of basic blocks in fs */
  long    fs_fsize;     /* size of frag blocks in fs */
  long    fs_frag;      /* number of frags in a block in fs */
/* these are configuration parameters */
  long    fs_minfree;   /* minimum percentage of free blocks */
  long    fs_rotdelay;  /* num of ms for optimal next block */
  long    fs_rps;       /* disk revolutions per second */
/* these fields can be computed from the others */
  long    fs_bmask;     /* ''blkoff'' calc of blk offsets */
  long    fs_fmask;     /* ''fragoff'' calc of frag offsets */
  long    fs_bshift;    /* ''lblkno'' calc of logical blkno */
  long    fs_fshift;    /* ''numfrags'' calc number of frags */
/* these are configuration parameters */
  long    fs_maxcontig; /* max number of contiguous blks */
  long    fs_maxbpg;    /* max number of blks per cyl group */
/* these fields can be computed from the others */
  long    fs_fragshift; /* block to frag shift */
  long    fs_fsbtodb;   /* fsbtodb and dbtofsb shift constant */
  long    fs_sbsize;    /* actual size of super block */
  long    fs_csmask;    /* csum block offset */
```

```
    long    fs_csshift;   /* csum block number */
    long    fs_nindir;    /* value of NINDIR */
    long    fs_inopb;     /* value of INOPB */
    long    fs_nspf;      /* value of NSPF */
/* yet another configuration parameter */
    long    fs_optim;     /* optimization preference, see below */
/* these fields are derived from the hardware */
    long    fs_npsect;    /* # sectors/track including spares */
    long    fs_interleave;/* hardware sector interleave */
    long    fs_trackskew; /* sector 0 skew, per track */
    long    fs_headswitch;/* head switch time, usec */
    long    fs_trkseek;   /* track-to-track seek, usec */
/* sizes determined by number of cylinder groups and sizes */
    daddr_t fs_csaddr;    /* blk addr of cyl grp summary area */
    long    fs_cssize;    /* size of cyl grp summary area */
    long    fs_cgsize;    /* cylinder group size */
/* these fields should be derived from the hardware */
    long    fs_ntrak;     /* tracks per cylinder */
    long    fs_nsect;     /* sectors per track */
    long      fs_spc;     /* sectors per cylinder */
/* this comes from the disk driver partitioning */
    long    fs_ncyl;      /* cylinders in file system */
/* these fields can be computed from the others */
    long    fs_cpg;       /* cylinders per group */
    long    fs_ipg;       /* inodes per group */
    long    fs_fpg;       /* blocks per group fs_frag */
/* this data must be re-computed after crashes */
    struct  csum fs_cstotal;
                          /* cylinder summary information */
/* these fields are cleared at mount time */
    char    fs_fmod;      /* super block modified flag */
    char    fs_clean;     /* file system is clean flag */
    char    fs_ronly;     /* mounted read-only flag */
    char    fs_flags;     /* currently unused flag */
    char    fs_fsmnt[MAXMNTLEN];
                          /* name mounted on */
    char    fs_fname[6];  /* file system name */
    char    fs_fpack[6];  /* file system back name */
/* these fields retain the current block allocation info */
    long  fs_cgrotor;     /* last cg searched */
    struct  csum *fs_csp[MAXCSBUFS];
                          /* list of fs_cs info buffers */
    long    fs_cpc;       /* cyl per cycle in postbl */
    short   fs_opostbl[16][8];
                          /* old rotation block list */
    long    fs_sparecon[53];
                          /* reserved for future constants */
    long    fs_id[2];     /* file system ID */
    long    fs_state;     /* file system state */
    quad    fs_qbmask;    /* ~fs_bmask - for use with quad size */
    quad    fs_qfmask;    /* ~fs_fmask - for use with quad size */
```

```
  long    fs_postblformat;
                        /* format of positional layout tables */
  long    fs_nrpos;     /* number of rotaional positions */
  long    fs_postbloff; /* (short) rotation block list head */
  long    fs_rotbloff;  /* (unchar) blocks for each rotation */
  long    fs_magic;     /* magic number */
  unchar  fs_space[1];  /* list of blocks for each rotation */
/* actually longer */
};
```

Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A file system is described by its superblock, which in turn describes the cylinder groups. The superblock is critical data and is replicated in each cylinder group to protect against catastrophic loss. This is done at file system creation time and the critical superblock data does not change, so the copies need not be referenced further unless disaster strikes.

Addresses stored in inodes are capable of addressing fragments of 'blocks'. File system blocks of at most size MAXBSIZE can be optionally broken into 2, 4, or 8 pieces, each of which is addressable; these pieces may be DEV_BSIZE, or some multiple of a DEV_BSIZE unit.

Large files consist of exclusively large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated as only as many fragments of a large block as are necessary. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determinable from information in the inode, using the "blksize(fs, ip, lbn)" macro.

The file system records space availability at the fragment level; to determine block availability, aligned fragments are examined.

The root inode is the root of the file system. Inode 0 cannot be used for normal purposes and historically bad blocks were linked to inode 1, thus the root inode is 2 (inode 1 is no longer used for this purpose, however numerous dump tapes make this assumption, so it remains). The *lost+found* directory is given the next available inode when it is initially created by **mkfs**.

fs_minfree gives the minimum acceptable percentage of file system blocks which may be free. If the freelist drops below this level only the superuser may continue to allocate blocks. *fs_minfree* may be set to 0 if no reserve of free blocks is deemed necessary, however severe performance degradations will be observed if the file system is run at greater than 90% full; thus the default value of *fs_minfree* is 10%.

The best trade-off between block fragmentation and overall disk utilitization at a loading of 90% comes with a fragmentation of 8, thus the default fragment size is an eighth of the block size.

Cylinder group related limits: Each cylinder keeps track of the availability of blocks at different rotational positions, so that sequential blocks can be laid out with minimum rotational latency. With the default of 8 distinguished rotation positions, the resolution of the summary information is 2ms for a typical 3600 rpm drive.

fs_rotdelay gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for *fs_rotdelay* is 2ms.

Each file system has a statically allocated number of inodes. An inode is allocated for each NBPI bytes of disk space. The inode allocation strategy is extremely conservative.

MINBSIZE is the smallest allowable block size. With a MINBSIZE of 4096 it is possible to create files of size 2^32 with only two levels of indirection. MINBSIZE must be big enough to hold a cylinder group block, thus changes to (struct cg) must keep its size within MINBSIZE. Note that super blocks are never more than size SBSIZE.

The path name on which the file system is mounted is maintained in *fs_fsmnt*. MAXMNTLEN defines the amount of space allocated in the super block for this name. The limit on the amount of summary information per file system is defined by MAXCSBUFS. For a 4096 byte block size, it is currently parameterized for a maximum of two million cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from *fs_csaddr* (size *fs_cssize*) in addition to the super block.

Note that *sizeof* (struct csum) must be a power of two in order for the "fs_cs" macro to work.

Super block for a file system: The size of the rotational layout tables is limited by the fact that the super block is of size SBSIZE. The size of these tables is **inversely** proportional to the block size of the file system. The size of the tables is increased when sector sizes are not powers of two, as this increases the number of cylinders included before the rotational pattern repeats (*fs_cpc*). The size of the rotational layout tables is derived from the number of bytes remaining in (struct fs).

The number of blocks of data per cylinder group is limited by the fact that cylinder groups are at most one block. The inode and free block tables must fit into a single block after deductin gspace for the cylinder group structure (struct cg).

Inode: The inode is the focus of all file activity in the file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is 'named' by its device/i-number pair. For further information, see the include file *<sys/ufsinode.h>*.

NAME

fspec – format specification in text files

DESCRIPTION

It is sometimes convenient to maintain text files on the operating system with non-standard tabs (tabs that are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by system commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets **<:** and **:>**. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

t*tabs* The **t** parameter specifies the tab settings for the file. The value of *tabs* must be one of the following:

1. a list of column numbers separated by commas, indicating tabs set at the specified columns;
2. a – followed immediately by an integer *n*, indicating tabs at intervals of *n* columns;
3. a – followed by the name of a "canned" tab specification.

Standard tabs are specified by **t–8**, which is equal to **t1, 9, 17, 25,** and so forth. Canned tabs that are recognized are defined by the **tabs**(1) command.

s*size* The **s** parameter specifies a maximum line size. The value of *size* must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.

m*margin* The **m** parameter specifies a number of spaces to be prepended to each line. The value of *margin* must be an integer.

d The **d** parameter takes no arguments. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

e The **e** parameter takes no arguments. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are **t–8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

SEE ALSO

ed(1), newform(1), tabs(1)

NAME

fstab – file-system-table

DESCRIPTION

The */etc/fstab* file contains information about file systems for use by **mount**(1M) and **mountall**(1M). Each entry in */etc/fstab* has the following format. The *mnt_** names refer to the *mntent* structure used by **getmntent**(3SEQ).

column 1	mnt_fsname	block special file name of file system or remote resource
column 2	mnt_dir	mount-point directory
column 3	mnt_type	file system type string
column 4	mnt_opts	options (comma separated no white space)
column 5+		ignored

White-space separates columns. Lines beginning with # are comments and empty lines are ignored.

The file system type string (column 3) determines how the file system and options fields (columns 1 and 4) are interpreted. A list of the filesystem types currently supported follows, including the way each type interprets these fields:

4.2 – *mnt_fsname* (column 1) must be a block special device.

ignore – The entire entry is ignored. This is useful to show disk partitions not currently used.

The options field (column 4) contains a list of comma-separated option words. Some option words are valid for all filesystem types, while others apply to a specific type only.

Option words valid on all file systems are as follows:

rw – read/write

ro – read-only

Note that either **rw** or **ro** (but not both) must be specified.

Option words specific to **4.2** file systems are as follows:

quota – usage limits enforced

noquota – usage limits not enforced

A file-system-table might read:

```
/dev/dsk/zd0s0   /          4.2  rw
/dev/dsk/zd0s7   /usr       4.2  rw,quota
/dev/dsk/zd1s7   /usr/src   4.2  ro
```

The order of records in */etc/fstab* is important because **mountall**(1M) processes the file sequentially; filesystems must appear after filesystems within which they are mounted.

The entries from this file are accessed using the routines in **getmntent**(3SEQ).

FILES

/etc/fstab

SEE ALSO

mount(1M), mountall(1M), mnttab(4), getmntent(3SEQ)

NAME

gettydefs – speed and terminal settings used by getty

DESCRIPTION

The */etc/gettydefs* file contains information used by **getty**(1M) to set up the speed and terminal settings for a line. It supplies information on what the **login**(1) prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by typing a break character.

Each entry in */etc/gettydefs* has the following format:

label# initial-flags # final-flags # login-prompt #next-label

Each entry is followed by a blank line. The various fields can contain quoted characters of the form **\b**, **\n**, **\c**, and so on, as well as *\nnn*, where *nnn* is the octal value of the desired character. The various fields are as follows:

label This is the string against which **getty**(1M) tries to match its second argument. It is often the speed (baud rate), such as 1200, at which the terminal is supposed to run, but it need not be (see below).

initial-flags These flags are the initial **ioctl**(2) settings to which the terminal is to be set if a terminal type is not specified to **getty**(1M). The flags that **getty**(1M) understands are the same as the ones listed in */usr/include/sys/termio.h* (see **termio**(7)). Normally only the speed flag is required in the *initial-flags*. **getty**(1M) automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-flag* settings remain in effect until **getty**(1M) executes **login**(1).

final-flags These flags take the same values as the *initial-flags* and are set just before **getty**(1M) executes **login**(1). The speed flag is again required. The composite flag SANE takes care of most of the other flags that need to be set so that the processor and terminal are communicating in a rational fashion. The other two commonly specified *final-flags* are TAB3, so that tabs are sent to the terminal as spaces, and HUPCL, so that the line is hung up on the final close.

login-prompt This entire field is printed as the *login-prompt*. Unlike the above fields where white space is ignored (a space, tab or newline), they are included in the *login-prompt* field.

next-label If this entry does not specify the desired speed, indicated by the user typing a break character, then **getty**(1M) searches for the entry with *next-label* as its *label* field and set up the terminal for those settings. Usually, a series of speeds are linked together in this fashion into a closed set; for instance, 2400 linked to 1200, which in turn is linked to 300, which finally is linked to 2400.

If **getty**(1M) is called without a second argument, then the first entry of */etc/gettydefs* is used, thus making the first entry of */etc/gettydefs* the default entry. It is also used if **getty**(1M) can not find the specified *label*. If */etc/gettydefs* itself is missing, there is one entry built into **getty**(1M) which brings up a terminal at **300** baud.

It is strongly recommended that after making or modifying */etc/gettydefs*, it be run through **getty**(1M) with the check option to be sure there are no errors.

FILES

/etc/gettydefs

SEE ALSO

login(1), stty(1), getty(1M), ioctl(2), termio(7)

WARNINGS

To support terminals that pass 8 bits to the system modify the entries in the */etc/gettydefs* file for those terminals by adding CS8 to *initial-flags* and replacing all occurrences of SANE with these values: BRKINT IGNPAR ICRNL IXON OPOST ONCLR CS8 ISIG ICANON ECHO ECHOK

An example of changing an entry in */etc/gettydefs* is illustrated below. All the information for an entry must be on one line in the file.

Original entry:

```
CONSOLE # B9600 HUPCL OPOST ONLCR # B9600 SANE IXANY
TAB3 HUPCL # Console Login:  # console
```

Modified entry:

```
CONSOLE # B9600 CS8 HUPCL OPOST ONLCR # B9600 BRKINT
IGNPAR ICNRL IXON OPOST ONLCR CS8 ISIG ICANON ECHO
ECHOK IXANY TAB3 HUPCL # Console Login:  # console
```

This change permits terminals to pass 8 bits to the system so long as the system is in multi-user state. When the system changes to single-user state, **getty**(1M) is killed and the terminal attributes are lost. Therefore to permit a terminal to pass 8 bits to the system in single-user state after entering single-user state, enter the following (see **stty**(1)):

stty –istrip cs8

NOTES

Customers who need to support terminals that pass 8 bits to the system (typical outside the U.S.A.) must modify the entries in */etc/gettydefs* as described in the **WARNINGS** section.

BUGS

8-bit with parity mode is not supported.

NAME

gps – graphical primitive string, format of graphical files

DESCRIPTION

A graphical primitive string (GPS) is a format used to store graphical data. Several routines have been developed to edit and display GPS files on various devices. Also, higher level graphics programs such as **plot** (in **stat**(1G)) and **vtoc** (in **toc**(1G)) produce GPS format output files.

GPS Primitives

A GPS is composed of five types of graphical data or primitives as follows:

lines The *lines* primitive has a variable number of points from which zero or more connected line segments are produced. The first point given produces a *move* to that location. (A *move* is a relocation of the graphic cursor without drawing.) Successive points produce line segments from the previous point. Parameters are available to set *color*, *weight*, and *style* (see below).

arc The *arc* primitive has a variable number of points to which a curve is fit. The first point produces a *move* to that point. If only two points are included, a line connecting the points will result; if three points are included, a circular arc through the points is drawn; and if more than three points are included, lines connect the points. (In the future, a spline will be fit to the points if they number greater than three.) Parameters are available to set *color*, *weight*, and *style*.

text The *text* primitive draws characters. It requires a single point which locates the center of the first character to be drawn. Parameters are *color*, *font*, *textsize*, and *textangle*.

hardware
The *hardware* primitive draws hardware characters or gives control commands to a hardware device. A single point locates the beginning location of the *hardware* string.

comment
A *comment* is an integer string that is included in a GPS file but causes nothing to be displayed. All GPS files begin with a comment of zero length.

GPS primitives are organized internally as follows:

lines *cw points sw*

arc *cw points sw*

text *cw point sw so* [*string*]

hardware
cw point [*string*]

comment
cw [*string*]

cw *cw* is the control word and begins all primitives. It consists of four bits that contain a primitive-type code and twelve bits that contain the word-count for that primitive.

point(s) *point(s)* is one or more pairs of integer coordinates. *text* and *hardware* primitives only require a single *point*. *point(s)* are values within a Cartesian plane or universe having 64K (–32K to +32K) points on each axis.

sw *sw* is the style-word and is used in *lines, arc,* and *text* primitives. For all three, eight bits contain *color* information. In *arc* and *lines*, eight bits are divided as four bits *weight* and four bits *style*. In the *text* primitive, eight bits of *sw* contain the *font*.

so *so* is the size-orientation word used in *text* primitives. Eight bits contain text size and eight bits contain text rotation.

string *string* is a null-terminated character string. If the string does not end on a word boundary, an additional null is added to the GPS file to insure word-boundary alignment.

GPS Parameters

The GPS parameters are as follows:

color *color* is an integer value set for *arc, lines,* and *text* primitives.

weight *weight* is an integer value set for *arc* and *lines* primitives to indicate line thickness. The value **0** is narrow weight, **1** is bold, and **2** is medium weight.

style *style* is an integer value set for *lines* and *arc* primitives to give one of the five different line styles that can be drawn on TEKTRONIX 4010 series storage tubes. They are:

- **0** solid
- **1** dotted
- **2** dot dashed
- **3** dashed
- **4** long dashed

font An integer value set for *text* primitives to designate the text font to be used in drawing a character string. (Currently *font* is expressed as a four-bit *weight* value followed by a four-bit *style* value.)

textsize *textsize* is an integer value used in *text* primitives to express the size of the characters to be drawn. *textsize* represents the height of characters in absolute *universe-units* and is stored at one-fifth this value in the size-orientation (*so*) word (see below).

textangle *textangle* is a signed integer value used in *text* primitives to express rotation of the character string around the beginning point. *textangle* is expressed in degrees from the positive x-axis and can be a positive or negative value. It is stored in the size-orientation (*so*) word as a value 256/360 of its absolute value.

SEE ALSO

graphics(1G), stat(1G), toc(1G)

NAME

group – group file

DESCRIPTION

group is an ASCII file containing the following information fields for each group:

group name
encrypted password
numerical group ID
comma-separated list of all users allowed in the group

The fields are separated by colons; each group is separated from the next by a new-line. If the password field is NULL, no password is required.

The **group** file resides in */etc*. Because the passwords are encrypted, the file has general read permission and, as an example, can be used to map numerical group ID's to names.

FILES

/etc/group

SEE ALSO

passwd(1), passwd(4), newgrp(1M)

NAME

helplog – monitor help actions

DESCRIPTION

The */usr/lib/help/HELPLOG* file contains a history of all actions taken by the **/usr/bin/help** command, if **help** monitoring has been enabled. The *HELPLOG* file is copied to */usr/lib/help/oHELPLOG* and a new */usr/lib/help/HELPLOG* file is created by the **/usr/lib/help/helpclean** command. Executing **helpclean** twice in succession zeros out the *HELPLOG* and the *oHELPLOG* files. Note that **/etc/cron, /etc/rc0**, or **/etc/rc2** can be used to clean up the **HELPLOG** file. To use one of these functions, add the appropriate command line to the */usr/spool/cron/crontab/root*, */etc/shutdown.d/* ..., or */etc/rc.d/ rc2.d, rc3.d* ... file, as applicable. The following command lines limit the size of the log file to the last 100 lines in the file:

```
tail -100 /usr/lib/help/HELPLOG > /tmp/help
mv /tmp/help /usr/lib/help/HELPLOG
```

EXAMPLES

The following display shows a typical */usr/lib/help/HELPLOG* file:

```
login=bin       uname=wr3b2a    date=Mon Aug 20 12:51:03 EDT 1984
name=locate     response='l'    status=OK
name=locate     response='d'    status=ERROR
name=getkey     response='k'    status=OK
name=keysrch    response='list' status=OK
name=quit       response='q'    status=OK
login=bin       uname=wr3b2a    date=Mon Aug 20 12:51:41 EDT 1984
```

SEE ALSO

cron(1M), rc(4)

NAME

init.d – system level transition files directory

DESCRIPTION

The */etc/init.d* directory contains executable files used in upward and downward transitions to all system run levels. These files are linked to files beginning with **S** (start) or **K** (stop) in **/etc/rc*n*.d**, where *n* is the appropriate run level. Files are not executed from this directory; they are only executed from **/etc/rc*n*.d** directories.

SEE ALSO

rc(4)

NAME

inittab – script for the init process

DESCRIPTION

The **inittab** file supplies the script to **init**(1M)'s role as a general process dispatcher. The process that constitutes the majority of **init**'s process dispatching activities is the line process */etc/getty* that initiates individual terminal lines. Other processes typically dispatched by **init** are daemons and the shell.

The **inittab** file is composed of entries that are position dependent and have the following format:

```
id:rstate:action:process
```

Each entry is delimited by a newline; a backslash (\) preceding a newline indicates a continuation of the entry. Up to 512 characters per entry are permitted. Comments may be inserted in the **process** field using the **sh**(1) convention for comments. Comments for lines that spawn **getty**s are displayed by the **who**(1) command. It is expected that they will contain some information about the line such as the location. There are no limits (other than maximum entry size) imposed on the number of entries within the **inittab** file. The entry fields are as follows:

id One to four characters used to uniquely identify an entry. Four character sequences are used for tty entries and should be avoided for other types of entries.

rstate The **run-level** in which this entry is to be processed. **run-levels** effectively correspond to a configuration of processes in the system. That is, each process spawned by **init** is assigned a **run-level** or **run-levels** in which it is allowed to exist. The **run-levels** are represented either by a number ranging from 0 through 6 or by the letters s or S (identical specifiers for single-user level). For a definition of each **run-level**, see **init** (1M).

As an example, if the system is in **run-level** 1, only those entries having a 1 in the **rstate** field will be processed. When **init** is requested to change **run-levels**, all processes which do not have an entry in the **rstate** field for the target **run-level** are sent the warning signal (SIGTERM) and allowed a 20-second grace period before being forcibly terminated by a kill signal (SIGKILL). The **rstate** field can define multiple **run-levels** for a process by selecting more than one **run-level** in any combination from 0–6. If no **run-level** is specified, then the process is assumed to be valid at all **run-levels** 0–6.

The level s (or S) can be used only with the initdefault, powerfail, powerwait and powerokwait action fields. Lines in *inittab* that contain s (or S) and do not have one of the above action fields will be ignored.

There are three other values, **a**, **b** and **c**, which can appear in the **rstate** field, even though they are not true **run-levels**. Entries which have these characters in the **rstate** field are processed only when the **telinit** (see **init**(1M)) process requests them to be run

(regardless of the current **run-level** of the system). These entries differ from **run-levels** in that **init** can never enter **run-level a, b** or **c**. Also, a request for the execution of any of these processes does not change the current **run-level**. Furthermore, a process started by an **a, b** or **c** command is not killed when **init** changes levels. They are only killed if their line in */etc/inittab* is marked **off** in the **action** field, their line is deleted entirely from */etc/inittab*, or **init** goes into the single-user state.

action Key words in this field tell **init** how to treat the process specified in the **process** field. The actions recognized by **init** are as follows:

respawn If the process does not exist, start the process; do not wait for its termination (continue scanning the **inittab** file), and when it dies restart the process. If the process currently exists, do nothing and continue scanning the **inittab** file.

wait Upon **init**'s entering the **run-level** that matches the entry's **rstate**, start the process and wait for its termination. All subsequent reads of the **inittab** file while **init** is in the same **run-level** cause **init** to ignore this entry.

once Upon **init**'s entering a **run-level** that matches the entry's **rstate**, start the process; do not wait for its termination. When it dies, do not restart the process. If upon entering a new **run-level**, where the process is still running from a previous **run-level** change, the program is not restarted.

boot The entry is to be processed only at **init**'s boot-time read of the **inittab** file. **init** is to start the process, not wait for its termination; and when it dies, not restart the process. In order for this instruction to be meaningful, the **rstate** should be the default or it must match **init**'s **run-level** at boot time. This action is useful for an initialization function following a hardware reboot of the system.

bootwait The entry is to be processed the first time **init** goes from single-user to multi-user state after the system is booted. (If **initdefault** is set to 2, the process runs right after the boot.) **init** starts the process, waits for its termination and, when it dies, does not restart the process.

powerfail Execute the process associated with this entry only when **init** receives a power-fail signal (SIGPWR) with a signal code value of PWR_FAIL. Refer to **signal**(2) and **sigcontext**(2seq). This action functions only on SSM-based systems.

powerwait Execute the process associated with this entry only when **init** receives a power fail signal (SIGPWR) with a signal code value of PWR_FAIL, and wait until it

terminates before continuing any processing of **inittab**. This action functions only on SSM-based systems.

powerokwait
Execute the process associated with this entry only when **init** receives a power fail signal (SIGPWR) with a signal code value of PWR_OK, and wait until it terminates before continuing any processing of **inittab**. This action functions only on SSM-based systems.

off If the process associated with this entry is currently running, send the warning signal (SIGTERM) and wait 20 seconds before forcibly terminating the process with the kill signal (SIGKILL). If the process is nonexistent, ignore the entry.

ondemand This instruction is really a synonym for the **respawn** action. It is functionally identical to **respawn** but is given a different keyword in order to divorce its association with **run-levels**. This is used only with the **a, b** or **c** values as described for the **rstate** field.

initdefault An entry with this **action** is only scanned when **init** is initially invoked. **init** uses this entry, if it exists, to determine which **run-level** to enter initially. It does this by taking the highest **run-level** specified in the **rstate** field and using that as its initial state. If the **rstate** field is empty, this is interpreted as 0123456, and so **init** will enter **run-level** 6. Additionally, if **init** does not find an **initdefault** entry in */etc/inittab*, it requests an initial **run-level** from the user at reboot time.

sysinit Entries of this type are executed before **init** tries to access the console (i.e., before the `Console Login:` prompt). It is expected that this entry will be only used to initialize devices on which **init** might try to ask the **run-level** question. These entries are executed and waited for before continuing.

process A **sh** command to be executed. The entire **process** field is prefixed with **exec** and passed to a forked **sh** as `sh -c 'exec` **command**'. For this reason, any legal **sh** syntax can appear in the **process** field. Comments can be inserted with the **;** *#comment* syntax.

FILES
/etc/inittab

SEE ALSO
sh(1), who(1), getty(1M), init(1M) exec(2), open(2), signal(2)

NAME

issue – issue identification file

DESCRIPTION

The file */etc/issue* contains the *issue* or project identification to be printed as a login prompt. This is an ASCII file which is read by program **getty** and then written to any terminal spawned or respawned from the *lines* file.

FILES

/etc/issue

SEE ALSO

login(1)

NAME

ldfcn – common object file access routines

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

DESCRIPTION

The common object file access routines are a collection of functions for reading common object files and archives containing common object files. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines insulate the calling program from knowledge of the overall structure of the object file.

The interface between the calling program and the object file access routines is based on the defined type LDFILE , defined as **struct ldfile**, declared in the header file *<ldfcn.h>*. The primary purpose of this structure is to provide uniform access to both simple object files and to object files that are members of an archive file.

The function **ldopen**(3X) allocates and initializes the LDFILE structure and returns a pointer to the structure to the calling program. The fields of the LDFILE structure may be accessed individually through macros defined in *<ldfcn.h>* and contain the following information:

LDFILE	`*ldptr;`
TYPE(ldptr)	The file magic number used to distinguish between archive members and simple object files.
IOPTR(ldptr)	The file pointer returned by **fopen** and used by the standard input-output functions.
OFFSET(ldptr)	The file address of the beginning of the object file; the offset is non-zero if the object file is a member of an archive file.
HEADER(ldptr)	The file header structure of the object file.

The object file access functions themselves may be divided into four categories:

(1) Functions that open or close an object file

ldopen(3X) and **ldaopen**[see **ldopen**(3X)]
open a common object file

ldclose(3X) and **ldaclose**[see **ldclose**(3X)]
close a common object file

(2) Functions that read header or symbol table information

ldahread(3X)
read the archive header of a member of an archive file

ldfhread(3X)
read the file header of a common object file

ldshread(3X) and **ldnshread**[see **ldshread**(3X)]
read a section header of a common object file

ldtbread(3X)
read a symbol table entry of a common object file

ldgetname(3X)
retrieve a symbol name from a symbol table entry or from the string table

(3) Functions that position an object file at (seek to) the start of the section, relocation, or line number information for a particular section.

ldohseek(3X)
seek to the optional file header of a common object file

ldsseek(3X) and **ldnsseek**[see **ldsseek**(3X)]
seek to a section of a common object file

ldrseek(3X) and **ldnrseek**[see **ldrseek**(3X)]
seek to the relocation information for a section of a common object file

ldlseek(3X) and **ldnlseek**[see **ldlseek**(3X)]
seek to the line number information for a section of a common object file

ldtbseek(3X)
seek to the symbol table of a common object file

(4) the function **ldtbindex**(3X) which returns the index of a particular common object file symbol table entry.

These functions are described in detail on their respective manual pages.

All the functions except **ldopen**(3X), **ldgetname**(3X), **ldtbindex**(3X) return either SUCCESS or FAILURE, both constants defined in *<ldfcn.h>*. Both **ldopen**(3X) and **ldaopen** (see **ldopen** (3X)) return pointers to an LDFILE structure.

Additional access to an object file is provided through a set of macros defined in *<ldfcn.h>*. These macros parallel the standard input-output file reading and manipulating functions, translating a reference of the LDFILE structure into a reference to its file descriptor field.

The following macros are provided:

GETC(ldptr)

FGETC(ldptr)

GETW(ldptr)

UNGETC(c, ldptr)

FGETS(s, n, ldptr)

FREAD((char *) ptr, sizeof (*ptr), nitems, ldptr)

FSEEK(ldptr, offset, ptrname)

FTELL(ldptr)

REWIND(ldptr)

FEOF(ldptr)

FERROR(ldptr)

FILENO(ldptr)

SETBUF(ldptr, buf)

STROFFSET(ldptr)

The STROFFSET macro calculates the address of the string table. See the manual entries for the corresponding standard input-output library functions for details on the use of the rest of the macros.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

fseek(3S), ldahread(3X), ldclose(3X), ldgetname(3X), ldfhread(3X), ldlread(3X), ldlseek(3X), ldohseek(3X), ldopen(3X), ldrseek(3X), ldsseek(3X), ldshread(3X), ldtbindex(3X), ldtbread(3X), ldtbseek(3X), stdio(3S), intro(5)

WARNINGS

The macro FSEEK defined in the header file *<ldfcn.h>* translates into a call to the standard input/output function **fseek**(3S). FSEEK should not be used to seek from the end of an archive file since the end of an archive file may not be the same as the end of one of its object file members.

NAME

limits – file header for implementation-specific constants

SYNOPSIS

#include <limits.h>

DESCRIPTION

The header file **<limits.h>** is a list of magnitude limitations imposed by a specific implementation of the operating system. All values are specified in decimal.

```
#define NGROUPS_MAX 16 /* # supplimentary GIDs per process */
#define ARG_MAX   12288/* max length of arguments to exec */
#define MAX_CANON 256  /* sizeof cononical input queue */
#define MAX_INPUT 256  /* # of bytes of atomic tty input */
#define CHAR_BIT  8    /* # of bits in a "char" */
#define CHAR_MAX  127  /* max integer value of a "char" */
#define CHAR_MIN  -128 /* min integer value of a "char" */
#define SCHAR_MIN -128 /* min integer value of a signed char */
#define SCHAR_MAX 127  /* max integer value of a signed char */
#define UCHAR_MAX 255  /* max integer value of an unsigned char */
#define CHILD_MAX 100  /* max # of processes per user ID */
#define CLK_TCK 100    /* # of clock ticks per second */
#define DBL_DIG 15     /* digits of precision of a "double" */
#define DBL_MAX 1.79769313486231470e+308
                       /*max decimal value of a "double" */
#define DBL_MIN 4.94065645841246544e-324
                       /*min decimal value of a "double" */
#define FCHR_MAX  1048576
                       /* max size of a file in bytes */
#define FLT_DIG 7      /* digits of precision of a "float" */
#define FLT_MAX 3.40282346638528860e+38
                       /*max decimal value of a "float" */
#define FLT_MIN 1.40129846432481707e-45
                       /*min decimal value of a "float" */
#define HUGE_VAL 3.40282346638528860e+38
                       /*error value returned by Math lib */
#define INT_MAX 2147483647
                       /* max decimal value of an "int" */
#define INT_MIN -2147483648
                       /* min decimal value of an "int" */
#define UINT_MAX 4294967295
                       /* max decimal value of an unsigned int */
#define ULONG_MAX 4294967295
                       /* max decimal value of an unsigned long */
#define USHRT_MAX 65535
                       /* max decimal value of an unsigned short */
#define LINK_MAX  1000
                       /* max # of links to a single file */
#define LONG_MAX  2147483647
                       /* max decimal value of a "long" */
#define LONG_MIN  -2147483648
```

```
                        /* min decimal value of a "long" */
#define MB_LEN_MAX  1   /* max bytes in a multibyte char */
#define NAME_MAX  255   /* max # of characters in a filename */
#define OPEN_MAX  20    /* max # of files a process can have open */
#define PASS_MAX  8     /* max # of characters in a password */
#define PATH_MAX  1023 /* max # of characters in a pathname */
#define PID_MAX 30000   /* max value for a process ID */
#define PIPE_BUF  8192 /* max # bytes atomic in write to a pipe */
#define PIPE_MAX  8192 /* max # bytes written to a pipe in a write */
#define SHRT_MAX  32767/* max decimal value of a "short" */
#define SHRT_MIN  -32768
                        /* min decimal value of a "short" */
#define STD_BLK 2048    /* # bytes in a physical I/O block */
#define SYS_NMLN  9     /* # of chars in uname-returned strings */
#define UID_MAX 60000   /* max value for a user or group ID */
#define USI_MAX 4294967295
                        /* max decimal value of an "unsigned" */
#define WORD_BIT  32    /* # of bits in a "word" or "int" */
#define LONG_BIT  32    /* # of bits in a long */
#define MAX_CHAR  256   /* max size of char input buffer */
#define SYSPID_MAX  30000
                        /* max pid of system processes */
#define TMP_MAX   17576/* max # of unique tmpnam names */
```

NAME

linenum – line number entries in a common object file

SYNOPSIS

#include <linenum.h>

DESCRIPTION

The **cc**(1) command, when invoked with the **–g** option, generates an entry in the object file for each C source line on which a breakpoint is possible. Users can then reference line numbers when using the appropriate software test system (see **pdbx**(1)). The structure of these line number entries is as follows:

```
struct      lineno
{
      union
      {
            long   l_symndx ;
            long   l_paddr ;
      }            l_addr ;
      unsigned shortl_lnno ;
} ;
```

Numbering starts with one for each function. The initial line number entry for a function has **l_lnno** equal to zero, and the symbol table index of the function's entry is in **l_symndx**. Otherwise, **l_lnno** is non-zero, and **l_paddr** is the physical address of the code for the referenced line. Thus the overall structure is as follows:

l_addr	**l_lnno**
function symtab index	0
physical address	line
physical address	line
...	
function symtab index	0
physical address	line
physical address	line
...	

SEE ALSO

cc(1), pdbx(1), a.out(4)

NAME

/usr/adm/loginlog – log of failed login attempts

DESCRIPTION

After five unsuccessful login attempts, all attempts are logged in the **loginlog** file. This file contains one record for each failed attempt. Each record contains the following information:

login name
tty specification
time

This is an ASCII file. Each entry is separated from the next by a new-line. Fields within each entry are colon-separated.

By default, **loginlog** does not exist, so no logging is done. To enable logging, the log file must be created with read and write permission for owner only. The owner must be **root** and the group must be **sys**.

FILES

/usr/adm/loginlog

SEE ALSO

login(1), passwd(1)

NAME

lptypes – table of printer types supported by the menu system

DESCRIPTION

The **lptypes** table contains a list of printer types that the Add a Printer command (**lpadmadd**) accepts.

FIELDS

Model Interface File Name

This is the name of the model interface file that should be substituted into the **lpadmin** command's **–m** option when the user selects the corresponding printer description. This file name must be executable and must exist in the */usr/spool/lp/model* directory.

Printer Description

This is the description of the printer type described by the model interface file. In most cases users are unable to understand the entries in the model interface file. This is a required field.

FILES

&{TFADMFILE}/etc/lptypes Table of printer types supported by the menu system

NOTES

Only the superuser (root) can update this file. All users can read this file.

NAME

mnttab – mounted filesystem table

DESCRIPTION

mnttab resides in directory */etc* and contains a table of filesystems currently mounted by the **mount**(1M) command. The **umount**(1M) command removes entries from this file.

The file contains a line of information for each mounted filesystem, structurally identical to the contents of */etc/fstab* as described in **fstab**(4), with the exception that columns 5 and beyond are replaced with a number representing the time (as returned by **time**(2)) when the filesystem was mounted.

The entries from the **mnttab** file are accessed using the routines in **getmntent**(3SEQ) or printed via **mount**(1M).

FILES

/etc/mnttab

SEE ALSO

mount(1M), setmnt(1M), umount(1M) getmntent(3SEQ), fstab(4)

NAME

mod_config – format of a module configuration file

DESCRIPTION

A **mod_config** file exists for each module in a kernel configuration file tree. The file describes what the module is and what other files are needed to generate the module. The directives below can appear in any order, with the exception of the **checkfor** directive which must appear as the first directive in a module configuration file when specified. Options can appear as a single quantity prepended with an equal sign, as a comma-separated list enclosed in parentheses, or as a line-delimited list enclosed in braces. Comments can appear preceded with a pound (#) sign. The directives are as follows:

Directive	Description
checkfor { *options* }	The named modules must be parsed before this one
block { *option* }	Block device major device number
character { *option* }	Character device major device number
streamd { *option* }	Stream driver major device number
streamm	Stream module
pseudo_device	Pseudo-device
filesystem { *option* }	File system type number
bus { *options* }	This module provides these busses
prefix { *option* }	Use this as the name prefix
maxpsuedos { *option* }	Maximum number of psuedos
maxunit { *option* }	Maximum permitted unit number
minunit { *option* }	Minimum permitted unit number
if*TYPE* (*parameter*) {	Conditional if*TYPE* (e.g. **ifdef**) TYPE is one of: **config, nconfig, def, ndef, product, nproduct.**
{ statements }	- *true clause*
} **elsif***TYPE* (*parameter*) {	Optional else if*TYPE* (e.g. **elsifdef**) TYPE is one of: **config, nconfig, def, ndef, product, nproduct.**
{ statements }	- *true clause*
} **else** {	Optional else
{ statements }	- *false clause*
}	
on_bus { *options* }	The module lives on these busses
syscalls { *option* }	This module provides this system call table
inits { *options* }	Declare these initialization functions
functions { *options* }	Declare these device driver functions
headers { *options* }	List of header files for this module

modules { *options* }	List of c and assembler files
space { *options* }	List of *space* files
makeopts { *options* }	Makefile options
conf { *options* }	*conf.c* options
confh { *options* }	*config.h* options
print { *option* }	Print argument
fatal { *option* }	Print argument and abort configuration
trace	Trace toggle
NEXT { *option* }	Next module definition
declare [*level*] { *params* }	Declare configurable parameters
default { *defaults* }	Define default parameter values
new_default { *defaults* }	Define new default parameter values
adjust { *param_exprs* }	Adjust parameter values
set { *param_exprs* }	Set parameter values
bound { *bound_exprs* }	Bound parameter values
assert { *assertions* }	Assert parameter values

The directive modifiers are as follows:

Modifier	**Description**
level	Defines parameter visibility level [1..10]
params	List of parameters in the form: *name type* { *"description"* }
defaults	List of default parameter values: *name* { *expr* } *name* { *expr* ; *bound* }
param_exprs	List of expressions adjusting or setting parameter values: *name* { *expr* }
bound_exprs	List of parameter bound expressions: *name* { *bound* }
assertions	List of parameter value assertions: { *expr relop expr* ; *"assert text"* }
bound	Bound interval for parameter: *low_bound_expr* .. .. *high_bound_expr* *low_bound_expr* .. *high_bound_expr*
expr	A simple arithmetic expression
relop	A relational operator: **<= < = > >=**

EXAMPLES

```
prefix="ex"
block=7
functions{"_init"}

space{ex_space.c}
modules {
        ixtr.c
        iytr.c
}
```

SEE ALSO

config (1m), sys_config (4)

NAME

motd – message-of-the-day file

DESCRIPTION

The */etc/motd* file contains the message-of-the-day. The message-of-the-day is displayed on your terminal by */etc/profile* after a successful login. This message should be kept short and to the point. The */usr/news* files should be used for lengthy, more explicit messages.

NAME

ott – files that hold object architecture information

DESCRIPTION

The FACE object architecture stores information about object types in an ASCII file named *.ott* (object type table) that is contained in each directory. This file describes all of the objects in that directory.

Each line of the *.ott* file contains information about one object in pipe-separated fields. The fields are in the following order:

name The name of the actual system file.

dname The name that should be displayed to the user, or a dot if it is the same as the name of the file.

description The description of the object, or a dot if the description is the default (the same as object-type).

object-type The FACE internal object type name.

flags Object specific flags.

mod time The time that FACE last modified the object. The time is given as number of seconds since 1/1/1970, and is in hexadecimal notation.

object information
An optional field, contains a set of semi-colon separated "name=value" fields that can be used by FACE to store any other information necessary to describe this object.

FILES

.ott created in any directory opened by FACE.

NAME

passwd – password file

DESCRIPTION

The **/etc/passwd** file identifes each user to the system. Entries are added for each new user. The **passwd** file is an ASCII file. Because user passwords are encrypted, the file has general read permission; therefore it can be used, for example, to map numerical user IDs to names. Entries for each user are separated by newlines. Fields within each user entry are separated by colons. */etc/passwd* contains the following information for each user:

login name:passwd:user:group:account:login directory:program

Explanations for these fields are as follows:

login name The first field defines the login name. The login name is from three to six characters long. The first character is alphabetic. The rest of the characters are alphanumeric. No uppercase characters are allowed.

passwd The password field consists of either an encrypted password, or the character **x** if there is an */etc/shadow* file. If */etc/shadow* does not exist, and the login does have a password, this field contains an encrypted copy of the password. This field remains only for compatibility reasons when **/etc/shadow** exists.

An encrypted login password contains 13 bytes (characters). The encrypted password consists of 13 characters chosen from a 64-character alphabet (., /, **0–9, A–Z, a–z**), except when the password is null, in which case the encrypted password is also null. The actual password is limited to a maximum of 8 bytes. The encrypted password can be followed by a comma and up to 4 more bytes of password aging information.

Password aging is in effect for a particular user if his encrypted password in the password file is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced in the first instance by the superuser.)

The first character of the age, *M*, for example, denotes the maximum number of weeks for which a password is valid. A user who attempts to login after his password has expired is forced to supply a new one. The next character, *m*, for example, denotes the minimum period in weeks which must expire before the password may be changed. The remaining one or two characters define the week (counted from the beginning of 1970) when the password was last changed. (A null string is equivalent to zero.) *M* and *m* have numerical values in the range 0–63 that correspond to the 64-character alphabet shown above (i.e., / = 1 week; z = 63 weeks). If $m = M = 0$ (derived from the string . or ..), the user is forced to change his password the next time he logs in (and the *age* disappears from his entry in the password file). If $m > M$ (signified, e.g., by the string *./*), only the superuser can change the password.

user id The third field contains the user identification number, which must be between 0 and 60,000. Group identification numbers 0 through 99 are reserved; 0 indicates the superuser (**root**). Commas are not allowed in this field.

group id The fourth field contains the group identification number, which must be between 0 and 60,000. Group identification numbers 0 through 99 are reserved; 0 indicates the superuser (**root**). Commas are not allowed in this field.

account The fifth field is used by accounting programs. This field typically contains the user name, department number, bin number, or other miscellaneous information.

login directory The sixth field defines the full pathname of the initial working directory (login directory).

program The seventh field defines the program to be executed after login. If this field is empty, the shell (**/bin/sh**) is invoked.

EXAMPLES

A typical portion of an **/etc/passwd** file follows:

```
root::0:1:0000-Admin(0000):/:
daemon:NP:1:1:0000-Admin(0000):/:
bin:NP:2:2:0000-Admin(0000):/bin:
sys:NP:3:3:0000-Admin(0000):/usr:
adm:NP:4:4:0000-Admin(0000):/usr/adm:
uucp:NP:5:5:0000-uucp(0000):/usr/lib/uucp:
nuucp:NP:10:10:0000-uucp(0000):/usr/spool/uucppublic:/usr/lib
sync::67:1:0000-Admin(0000):/:/bin/sync
lp:NP:71:2:0000-lp(0000):/usr/spool/lp:
usrlimit:NP:88:4:1 user system (DON'T CHANGE THIS LINE):/:/dev/null
setup::0:0:general system administration:/usr/admin:/usr/bin
powerdown:NP:0:0:general system administration:/usr/admin
sysadm:NP:0:0:general system administration:/usr/bin/sysadm
bupsched:NP:0:0:administration login:/usr/admin:/usr/bin/bupsched
checkfsys:NP:0:0:administration login:/usr/bin/checkfsys
mountfsys:NP:0:0:administration login:/usr/admin:/usr/bin/mountfsys
umountfsys:NP:0:0:administration login:/usr/bin/umountfsys
vmsys:NP:4000:400:FACE usr login:/usr/vmsys:
oasys:NP:4001:401:Object Architecture Files:/usr/oasys:
anneh:x:100:1:Anne Harris, Sales, 9238:/usr/anneh:
bobs:x:101:1:Robert Smith, Engineering, 1234:/usr/bobs:
coleenl:x:102:1:Coleen Laser:/usr/coleenl:
dol:x:103:1:David Osborne:/usr/dol:/bin/ksh
```

FILES

/etc/passwd
/etc/shadow

SEE ALSO

login(1), passwd(1), passmgmt(1M), getpwent(3C), group(4), shadow(4)

NAME

plot – graphics interface

DESCRIPTION

Files of this format are produced by routines described in **plot**(3X) and are interpreted for various devices by commands described in **tplot**(1G). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the *x* and *y* values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the "current point" for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in **plot**(3X):

- **m** move: The next four bytes give a new current point.
- **n** cont: Draw a line from the current point to the point given by the next four bytes (see **tplot**(1G)).
- **p** point: Plot the point given by the next four bytes.
- **l** line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.
- **t** label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a new-line.
- **e** erase: Start another frame of output.
- **f** linemod: Take the following string, up to a new-line, as the style for drawing further lines. The styles are *dotted, solid, longdashed, shortdashed,* and *dotdashed.* This is effective only for the **–T4014** and **–Tver** options of **tplot**(1G) (TEKTRONIX 4014 terminal and Versatec plotter).
- **s** space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of **tplot**(1G). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

DASI 300	space(0, 0, 4096, 4096);
DASI 300s	space(0, 0, 4096, 4096);
DASI 450	space(0, 0, 4096, 4096);
TEKTRONIX 4014	space(0, 0, 3120, 3120);
Versatec plotter	space(0, 0, 2048, 2048);

SEE ALSO

graph(1G), tplot(1G), plot(3X), gps(4), term(5)

WARNING

The plotting library, **plot**(3X), and the curses library, **curses**(3X), both use the names erase() and move(). The curses versions are macros. If you need both libraries, put the code in a different source file than the **curses**(3X) code, or

#undef move() and erase() in the **plot**(3X) code.

NAME

product – installed products file

DESCRIPTION

This file contains a record of all software products that have been installed using the product installation routines. The information in the file is used by the routines that back up and remove software products.

FIELDS

Product Code
This is the name of the product. When the installation procedure completes successfully, the system uses this name as a prefix for the file that is created: */usr/options/prod.mnt531*. This is a required field.

Product Class
This describes the type of product, such as Word Processor, Spreadsheet, or Database. You can sort the file by this field to list all products in a class. This is a required field.

Product Description
When the user selects the Run, Back up, or Remove options, this description appears in the SELECT window. This is a required field.

Binary Home Directory
The name of the parent directory where the product binaries are stored. This *may* be used by the product backup and/or removal routines. This is a required field.

Data Home Directory
The name of the parent directory that holds the product data. This *may* be used by the product backup and/or removal routines. This is a required field.

Version Number
This field holds the version number of the installed product. This is a required field.

Program to invoke
Name of the program that is initiated when the user selects this product from the Run Application SELECT window. This is a required field.

FILES

${TFADMFILE}/etc/product	Installed Products File

NOTES

This file is maintained by the System Administrator. If product installation and/or removal procedures are bypassed and products are added or removed manually, the information in this file may not be correct.

NAME

profile – setting up an environment at login time

SYNOPSIS

/etc/profile
$HOME/.profile

DESCRIPTION

Users who have the shell **sh**(1) as their login command have the commands in */etc/profile* and **$HOME/***.profile* executed as part of the login sequence.

/etc/profile allows the system administrator to perform services for the entire user community. Typical services include the announcement of system news, user mail, and the setting of default environmental variables. It is not unusual for */etc/profile* to execute special actions for the **root** login or the **su**(1) command. Computers running outside the Eastern time zone should have the following line included early in */etc/profile* (see **timezone**(4)).

```
. /etc/TIMEZONE
```

The file *$HOME/.profile* is used for setting exported environment variables and terminal modes specific to each user. A typical example (except for the comments) of a of a user's *.profile* file follows:

```
#  Make some environment variables global
export MAIL PATH TERM
#  Set file creation mask
umask 027
#  Tell me when new mail comes in
MAIL=/usr/mail/$LOGNAME
#  Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
#  Set terminal type
while :
do        echo "terminal: \c"
          read TERM
          if [ -f ${TERMINFO:-/usr/lib/terminfo}/?/$TERM ]
          then break
          elif [ -f /usr/lib/terminfo/?/$TERM ]
          then break
          else echo "invalid term $TERM" 1>&2
          fi
done
#  Initialize the terminal and set tabs
#  The environmental variable TERM must have been exported
#  before the "tput init" command is executed.
tput init
#  Set the erase character to backspace
stty erase '^H' echoe
```

FILES

/etc/TIMEZONE	Timezone environment
$HOME/.profile	User-specific environment
/etc/profile	System-wide environment

SEE ALSO

env(1), login(1), mail(1), sh(1), stty(1), su(1), tput(1), su(1M), terminfo(4), timezone(4), environ(5), term(5)

NOTES

Care must be taken in providing system-wide services in */etc/profile*. Personal *.profile* files are better for serving all but the most global needs.

NAME

proto – prototype job file for at

SYNOPSIS

/usr/lib/cron/.proto

/usr/lib/.proto*queue*

DESCRIPTION

When a job is submitted to **at**(1) or **batch**(1), the job is constructed as a shell script. First, a prologue is constructed, consisting of:

- A header whether the job is an **at** job or a **batch** job (actually, **at** jobs submitted to all queues other than queue **a**, not just to the batch queue **b**, are listed as **batch** jobs); the header is

  : at job

for an **at** job, and

: batch job

for a **batch** job.

- A set of Bourne shell commands to make the environment (see *environ*(5)) for the **at** job the same as the current environment;
- A command to run the user's shell (as specified by the **SHELL** environment variable) with the rest of the job file as input.

at then reads a *prototype file*, and constructs the rest of the job file from it.

Text from the prototype file is copied to the job file, except for special *variables* that are replaced by other text:

$d	is replaced by the current working directory
$l	is replaced by the current file size limit (see *ulimit*(2))
$m	is replaced by the current umask (see *umask*(2))
$t	is replaced by the time at which the job should be run, expressed as seconds since January 1, 1970, 00:00 Univeral Time Code (Greenwich Mean Time), preceded by a colon
$<	is replaced by text read by **at** from the standard input (that is, the commands provided to **at** to be run in the job)

If the job is submitted in queue *queue*, **at** uses the file **/usr/lib/cron/.proto***queue* as the prototype file if it exists, otherwise it will use the file **/usr/lib/cron/.proto**.

EXAMPLES

The standard **.proto** file supplied is:

```
#ident"@(#)adm:.proto    1.2"
cd $d
ulimit $l
umask $m
$<
```

which causes commands to change the current directory in the job to the current directory at the time **at** was run, to change the file size limit in the job to the file size limit at the time **at** was run, and to change the umask in the job to the umask at the time **at** was run, to be inserted before the commands in

the job.

FILES

/usr/lib/cron/.proto
/usr/lib/cron/.proto*.queue*

SEE ALSO

at (1)

NAME

rc – reboot and shutdown command scripts

DESCRIPTION

The following files and directories are described here:
/etc/rc0
/etc/rc0.d
/etc/rc2
/etc/rc2.d
/etc/rc.d
/etc/rc3

/etc/rc0 File

The */etc/rc0* file contains a shell script that is executed by **shutdown** for transitions to single-user state, and by **init** on transitions to run levels 0, 5, and 6. Files in the */etc/shutdown.d* and */etc/rc0.d* directories are executed when */etc/rc0* is run. The file *K00ANNOUNCE* in */etc/rc0.d* prints the message `System services are now being stopped.` Any task that you want executed when the system is taken to run levels 0, s, 5, or 6 can be done by adding a file to the */etc/shutdown.d* directory.

/etc/rc0.d Directory

The */etc/rc0.d* directory contains files executed by */etc/rc0* for transitions to system run levels 0, 5, and 6. Files in this directory are linked from the */etc/init.d* directory, and begin with either a **K** or an **S**. **K** indicates processes that are stopped, and **S** indicates processes that are started when entering run levels 0, 5, or 6.

/etc/rc2 File

The */etc/rc2* file is a shell script that is executed by **/etc/init** on transitions to run level 2 (multi-user state). Executable files in the */etc/rc.d* directory and any executable files beginning with **S** or **K** in */etc/rc2.d* directories are executed when */etc/rc2* is run. All files in *rc2.d* are linked from files in the */etc/init.d* directory. The following are descriptions of some of those files. You can add your own custom files to the */etc/rc2.d* and */etc/rc.d* directories if you add hardware or software to the system (see **rc2**(1M)). These files are prefixed with an **S** or a **K** and a number in the */etc/rc2.d* directory.

MOUNTFSYS Sets up and mounts all filesystems listed in */etc/fstab* by running the */etc/mountall* program. After all filesystems are mounted, **MOUNTFSYS** runs **quotacheck**(1M) and **quotaon(1M)** for filesystems that have quotas enabled (see **fstab**(4)). If there is a problem mounting any filesystem, a warning is issued and the system is returned to single-user mode (see **rc2**(1M).

PRESERVE Recover editing sessions that may have otherwise been lost in the event of a power loss.

RMTMPFILES Makes the */tmp* and */usr/tmp* directories, cleaning up (deleting) any previous files in that directory.

addswap Enable multiple disk partitions to be configured as swap devices.

bootflags	Sets up the system specific boot information using the **/etc/bootflags** command.
cron	Starts the *cron* daemon by executing */etc/cron*.
savecore	Saves a system memory dump into the /usr/crash directory.
sysetup	Removes */etc/ps_data* to force the */bin/ps* command to read the */unix* file. Outputs the system trademark information. Puts the version string of the booted kernel in */etc/motd*.
uucp	When basic networking is added to the system, the *uucp* file is added to this directory. The *uucp* file deletes *uucp* locks (LCK*), status files (STST*), and temporary files (TM*) under the */usr/spool/uucp* directory structure.
lp	The *lp* file removes the spooler lock file and starts the printer scheduler on startup, and stops the scheduler on shutdown.

/etc/rc2.d Directory

The */etc/rc2.d* directory contains files executed by */etc/rc2* for transitions to system run level 3. Files in this directory are linked from the */etc/init.d* directory, and begin with either a **K** or an **S**. **K** indicates processes that should be stopped, and **S** indicates processes that should be started when entering run levels 2 or 3.

/etc/rc.d Directory

The */etc/rc.d* directory contains executable files that do the various functions needed to initialize the system to run level 2. The files are executed when */etc/rc2* is run. This directory is maintained only for backward compatibility.

/etc/rc3 File

The */etc/rc3* file is executed by */etc/init* and. could contain the various functions needed to initialize the system to run level 3. Run level 3 is a superset of run level 2, in that */etc/rc3* will execute */etc/rc2* if the transition was from a run level less than 2.

/etc/rc3.d Directory

The */etc/rc3.d* directory contains files executed by */etc/rc3* for transitions to system run level 3 (multi-user mode). Files in this directory are linked from the **/etc/init.d** directory, and begin with either a **K** or an **S**. **K** indicates processes that should be stopped, and **S** indicates processes that should be started when entering run level 3.

EXAMPLES

The following shows a typical */etc/rc0* file:

```
#! /bin/sh
#       "Run Commands" for init state 0
#       Leaves the system in a state where it
#       it is safe to turn off the power or go
```

```
#       to firmware.


echo 'The system is coming down.  Please wait.'

#       The following segment is for historical purposes.
#       There should be nothing in /etc/shutdown.d.
if [ -d /etc/shutdown.d ]
then
        for f in /etc/shutdown.d/*
        {
                if [ -s $f ]
                then
                        /bin/sh ${f}
                fi
        }
fi
#       End of historical section

if [ -d /etc/rc0.d ]
then
        for f in /etc/rc0.d/K*
        {
                if [ -s ${f} ]
                then
                        /bin/sh ${f} stop
                fi
        }

#       system cleanup functions ONLY (things that end fast!)

        for f in /etc/rc0.d/S*
        {
                if [ -s ${f} ]
                then
                        /bin/sh ${f}' start
                fi
        }
fi

trap "" 15
kill -15 -1
sleep 5
/etc/killall  9
sync;sync;sync
/etc/umountall
sync;  sync
echo '
The system is down.'
sync
uadmin 2 0
```

The following shows a typical /etc/rc2 file:

```
#! /bin/sh
#       "Run Commands" executed when the system
#       is changing to init state 2,  traditionally
#       called "multi-user".


#       Pickup start-up packages for mounts,
#         daemons, services, etc.
set `who -r`
if [ $9 = "S" ]
then
        echo 'The system is coming up.  Please wait.'
        BOOT=yes
        if [ -f /etc/rc.d/PRESERVE ]    # historical segment
        then                            # for vi and ex
                mv /etc/rc.d/PRESERVE /etc/init.d
                ln /etc/init.d/PRESERVE /etc/rc2.d/S02PRESERVE
        fi

elif [ $7 = "2" ]
then
        echo 'Changing to state 2.'
        if [ -d /etc/rc2.d ]
        then
                for f in /etc/rc2.d/K*
                {
                        if [ -s ${f} ]
                        then
                                /bin/sh ${f} stop
                        fi
                }
        fi
fi

# If the .single_user file gets created, it will contain
# the reason to avoid going multi-user.  Send init the
# SINGLE_USER signal, print out the reason for going back
# to single user mode and exit.  Init cannot proceed with
# the run-level change until this script exits.
/bin/rm -f /etc/rc2.d/.single_user
if [ -d /etc/rc2.d ]
then
        for f in /etc/rc2.d/S*
        {
                if [ -s ${f} ]
                then
                        /bin/sh ${f} start
                        status=$?
                        if [ -f /etc/rc2.d/.single_user ]
                        then
```

```
                                        /etc/init S
                                        echo '*** Going back to SINGLE
                                                    USER mode!'
                                        /bin/cat /etc/rc2.d/.single_user
                                        exit $status
                                fi
                        fi
                }
        fi

        if [ "${BOOT}" = "yes" -a -d /etc/rc.d ]
        then
                for f in `ls /etc/rc.d`
                {
                        if [ ! -s /etc/init.d/${f} ]
                        then
                                /bin/sh /etc/rc.d/${f}
                        fi
                }
        fi

        if [ "${BOOT}" = "yes" -a $7 = "2" ]
        then
                echo 'The system is ready.'
        elif [ $7 = "2" ]
        then
                echo 'Change to state 2 has been completed.'
        fi
```

SEE ALSO

shutdown(1), rc0(1M), rc2(1M)

NAME

rcsfile – format of RCS file

DESCRIPTION

An RCS file is an ASCII file. Its contents are described by the grammar below. The text is free format; spaces, tabs and new lines have no significance except in strings. Strings are enclosed by @. If a string contains a @, it must be doubled.

The meta syntax uses the following conventions: | (bar) separates alternatives; { and } (braces) enclose optional phrases; { and }* enclose phrases that may be repeated zero or more times; { and }+ enclose phrases that must appear at least once and may be repeated; and < and > enclose nonterminals.

<rcstext>	::=	<admin> {<delta>}* <desc> {<deltatext>}*	
<admin>	::=	**head**	{<num>};
		branch	{<num>};
		access	{<id>}*;
		symbols	{<id> : <num>}*;
		locks	{<id> : <num>}*;
		comment	{<string>};
<delta>	::=	<num>	
		date	<num>;
		author	<id>;
		state	{<id>};
		branches	{<num>}*;
		next	{<num>};
<desc>	::=	**desc**	<string>
<deltatext>	::=	<num>	
		log	<string>
		text	<string>
<num>	::=	{<digit>{.}}+	
<digit>	::=	0 | 1 | ... | 9	
<id>	::=	<letter>{<idchar>}*	
<letter>	::=	A | B | ... | Z | a | b | ... | z	
<idchar>	::=	Any printing ASCII character except space, tab, carriage return, new line, and <special>.	
<special>	::=	; | : | , | @	
<string>	::=	@{any ASCII character, with '@' doubled}*@	

Identifiers are case sensitive. Keywords are in lowercase only. The sets of keywords and identifiers may overlap.

The <delta> nodes form a tree. All nodes whose numbers consist of a single pair (for example 2.3, 2.1, 1.3) are on the trunk of the tree, and are linked through the **next** field in order of decreasing numbers. The **head** field in the <admin> node points to the head of that sequence (that is, it contains the highest pair). The **branch** node in the admin node indicates the default branch (or revision) for most RCS operations. If empty, the default branch is the highest branch on the trunk.

All <delta> nodes whose numbers consist of 2n fields (n≥2) (for example 3.1.1.1, 2.1.2.2) are linked in the following way.

> All nodes whose first (2n)-1 number fields are identical are linked through the **next** field in order of increasing numbers. For each such sequence, the <delta> node whose number is identical to the first 2(n-1) number fields of the deltas on that sequence is called the branchpoint. The **branches**111eld of a node contains a list of the numbers of the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers.

SEE ALSO

ci(1), co(1), ident(1), rcs(1), rcsdiff(1), rcsmerge(1), rlog(1)

NAME

reloc – relocation information for a common object file

SYNOPSIS

#include <reloc.h>

DESCRIPTION

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it is in the following format:

```
struct reloc
{
        long    r_vaddr ; /* (virtual) address of reference */
        long    r_symndx ;/* index into symbol table */
        ushort  r_type ;  /* relocation type */
} ;

#define         R_ABS     0
```

As the link editor, **ld**(1), reads each input section and performs relocation, the relocation entries are read. The relocation entries direct how references found within the input section are treated, as follows:

R_ABS The reference is absolute and no relocation is necessary. The entry will be ignored.

R_DIR16 A direct 16-bit reference to a symbol's virtual address.

R_REL16 A "PC-relative" 16-bit reference to a symbol's virtual address. The actual address is calculated by adding a constant to the PC value.

R_IND16 A 16-bit reference through a transfer vector table.

R_OFF8 An 8-bit offset reference in 8 bits.

R_OFF16 An 8-bit offset reference in 16 bits.

R_SEG12 A 12-bit segment reference.

R_AUX An auxiliary relocation entry.

More relocation types exist for other processors. Equivalent relocation types on different processors have equal values and meanings. New relocation types will be defined (with new values) as they are needed.

Relocation entries are generated automatically by the assembler (**as** (1)) and automatically used by the link editor. Link editor options exist for both preserving and removing the relocation entries from object files.

SEE ALSO

as(1), ld(1), a.out(4), syms(4)

NAME

sccsfile – format of SCCS file

DESCRIPTION

An SCCS (Source Code Control System) file is an ASCII file with six logical parts:

the checksum,

the delta table
(contains information about each delta),

user names
(contains login names and/or numerical group IDs of users who may add deltas),

flags (contains definitions of internal keywords),

comments
(contains arbitrary descriptive information about the file),

the body (contains the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is referred to here as the control character and is represented graphically as @.

Entries of the form **DDDDD** represent a five-digit string (a number between 00000 and 99999).

Each logical part of an SCCS file is described in detail below.

Checksum

The checksum is the first line of an SCCS file. The form of the line is as follows:

@hDDDDD

The value of the checksum is the sum of all characters, except those of the first line. The **@h** provides a magic number of (octal) 064001.

Delta table

The delta table consists of a variable number of entries of the form:

@s DDDDD/DDDDD/DDDDD
@d <type> <SCCS ID> yr/mo/da hr:mi:se <pgmr> **DDDDD DDDDD**
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
.
.
.
@c <comments> ...
.
.
.
@e

The first line (**@s**) contains the number of lines inserted/deleted/unchanged, respectively. The second line (**@d**) contains the type of the delta (currently, normal: **D**, and removed: **R**), the SCCS ID of the delta, the date and time of creation of the delta, the
login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The **@i**, **@x**, and **@g** lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

The **@m** lines (optional) each contain one MR number associated with the delta; the **@c** lines contain comments associated with the delta.

The **@e** line ends the delta table entry.

User names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines **@u** and **@U**. An empty list allows anyone to make a delta. Any line starting with a **!** prohibits the succeeding group or user from making deltas.

Flags

Keywords used internally. (See **admin**(1) for more information on their use.) Each flag line takes the form:

@f <flag> <optional text>

The following flags are defined:

@f t	<type of program>
@f v	<program name>
@f i	<keyword string>
@f b	
@f m	<module name>
@f f	<floor>
@f c	<ceiling>
@f d	<default-sid>
@f n	
@f j	
@f l	<lock-releases>
@f q	<user defined>
@f z	<reserved for use in interfaces>

The **t** flag defines the replacement for the **%Y%** identification keyword. The **v** flag controls prompting for **MR** numbers in addition to comments; if the optional text is present it defines an **MR** number validity checking program. The **i** flag controls the warning/error aspect of the "No id keywords" message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message causes a fatal error (the file is not gotten, or the delta is not made). When the **b** flag is present the **–b** keyletter may be used on the **get** command to cause a branch in the delta tree. The **m** flag defines the first choice for the replacement text of the **%M%** identification keyword. The **f** flag

defines the "floor" release; the release below which no deltas may be added. The **c** flag defines the "ceiling" release; the release above which no deltas may be added. The **d** flag defines the default SID to be used when none is specified on a **get** command. The **n** flag causes **delta** to insert a "null" delta (a delta that applies no changes) in those releases that are skipped when a delta is made in a new release (e.g., when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty. The **j** flag causes **get** to allow concurrent edits of the same base SID. The **l** flag defines a *list* of releases that are locked against editing [**get**(1) with the **-e** keyletter]. The **q** flag defines the replacement for the **%Q%** identification keyword. The **z** flag is used in certain specialized interface programs.

Comments

Arbitrary text is surrounded by the bracketing lines **@t** and **@T**. The comments section typically will contain a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines do not begin with the control character, control lines do. There are three kinds of control lines: insert, delete, and end, represented by the following, respectively:

@I DDDDD
@D DDDDD
@E DDDDD

The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

admin(1), delta(1), get(1), prs(1)

NAME

scnhdr – section header for a common object file

SYNOPSIS

#include <scnhdr.h>

DESCRIPTION

Every common object file has a table of section headers to specify the layout of the data within the file. Each section within an object file has its own header. The C structure appears below.

```
struct scnhdr
{
  char     s_name[SYMNMLEN];     /* section name */
  long     s_paddr;              /* physical address */
  long     s_vaddr;              /* virtual address */
  long     s_size;               /* section size */
  long     s_scnptr;             /* file ptr to raw data */
  long     s_relptr;             /* file ptr to relocation */
  long     s_lnnoptr;            /* file ptr to line numbers */
  unsigned short  s_nreloc;      /* # reloc entries */
  unsigned short  s_nlnno;       /* # line number entries */
  long     s_flags;              /* flags */
} ;
```

File pointers are byte offsets into the file; they can be used as the offset in a call to **FSEEK** (see **ldfcn**(4)). If a section is initialized, the file contains the actual bytes. An uninitialized section has a size, symbols defined in it, and symbols that refer to it; but, it can have no relocation entries, line numbers, or data. Consequently, an uninitialized section has no raw data in the object file, and the values for **s_scnptr**, **s_relptr**, **s_lnnoptr**, **s_nreloc**, and **s_nlnno** are zero.

SEE ALSO

ld(1), fseek(3S), a.out(4)

NAME

scr_dump – format of curses screen image file

SYNOPSIS

scr_dump(file)

DESCRIPTION

The **curses**(3X) function **scr_dump()** copies the contents of the screen into a file. The format of the screen image is described below.

The name of the *tty* is 20 characters long and the modification time (the **mtime** of the *tty* that this is an image of) is of the type **time_t**. All other numbers and characters are stored as **chtype** (see **<curses.h>**). No newlines are stored between fields. The screen image format is as follows:

```
<magic number: octal 0433>
<name of tty>
<mod time of tty>
<columns> <lines>
<line length> <chars in line>          for each line on the screen
<line length> <chars in line>
  .
  .
  .
<labels?>                              1, if soft screen labels are present
<cursor row> <cursor column>
```

Only as many characters as are in a line are listed. For example, if the <line length> is 0, there will be no characters following <line length>. If <labels?> is TRUE, following it will be

```
<number of labels>
<label width>
<chars in label 1>
<chars in label 2>
  .
  .
  .
```

SEE ALSO

curses(3X)

NAME

shadow – restricted password file

DESCRIPTION

To protect encrypted user passwords, an optional security feature allows a system administrator to move all password and aging information from the publicly-readable */etc/passwd* file to an access-restricted file called the *shadow* password file. The *shadow* password file contains one entry per login (user). Each entry consists of the following fields that are colon-separated and in the following form:

username:password:lastchanged:min:max

username The user's login name (ID).

password A 13-character encrypted password for the user or a *lock* string to indicate that the login is not accessible; or no string which indicates that there is no password for the login.

lastchanged The number of days between January 1, 1970, and the date that the password was last modified.

min The minimum number of days required between password changes.

max The maximum number of days the password is valid.

EXAMPLES

A typical portion of a *shadow* password file follows. Notice the lock string NONE in the *bin, sys*, and *adm* entries.

```
root::6445::
daemon::6445::
bin:NONE:6445::
sys:NONE:6445::
adm:NONE:6445::
uucp::6445::
nuucp::6445::
sync::6445::
lp::6445::
usrlimit::7069::
setup::6445::
powerdown::6445::
sysadm::6445::
bupsched::6445::
checkfsys::6445::
mountfsys::6445::
umountfsys::6445::
vmsys::6445::
oasys::6445::
anneh:djw843Kehf73s:6445:90:120
bobs:x:Iod3iRzt7eRK4:6445:90:120
coleenl:lc9gjllVs8xU9:6445:90:120
dol:mCi9r46jTY9uI:6445:90:120
```

SEE ALSO

passwd(1), passmgmt(1M), passwd(4)

NAME

shells – shell table

DESCRIPTION

The shell table contains a list of initial programs that the "Add a User Account" **(addusr)** command accepts.

FIELDS

Shell Description

This is the description of the shell program. The description is displayed in the SELECT window for the shell field in the **addusr** form. This is a required field.

Shell Executable

The name of the binary program described in the first field. The program must be on disk and should be executable by all users. This value is copied to the **shell** input line when the user selects the corresponding description. This is a required field.

SEE ALSO

menu(1M)

NOTES

Only the superuser (root) can update this file. All users can read this file.

NAME

shutdown – system shutdown information file

DESCRIPTION

The **/etc/shutdown** file contains a shell script to shut down the system gracefully in preparation for system backup or scheduled downtime. After stopping all nonessential processes, the **shutdown** script executes files in the **/etc/shutdown.d** directory by calling **/etc/rc0** for transitions to run level s or S. For transitions to other run levels, the **shutdown** script calls **/etc/init**. A typical **/etc/shutdown** file follows:

```
# Sequence performed to change the init stat of a machine.
# This procedure checks to see if you are permitted and
# allows an interactive shutdown.  The actual change of
# state, killing of processes and such are performed by
# the new init state, say 0, and its /etc/rc0.
#
# Usage:
#  shutdown [ -y ] [ -g<grace-period> ] [ -i<init-state> ]
#
#!     chmod +x ${file}
#
#
if [ `pwd` != / ]
then
echo "$0:  You must be in the / directory to run
                                    /etc/shutdown."
exit 1
fi
#     Check the user id.
if [ -x /usr/bin/id ]
then
      eval `id  |  sed 's/[^a-z0-9=].*//'`
      if [ "${uid:=0}" -ne 0 ]
      then
            echo "$0: Only root can run /etc/shutdown."
          exit 2
      fi
fi
grace=60
askconfirmation=yes
initstate=s
while [ $# -gt 0 ]
do
      case $1 in
      -g[0-9]* )
            grace=`expr "$1" : '-g)'`
            ;;
      -i[Ss0156] )
      initstate=`expr "$1" : '-i)'`
      ;;
```

```
    -i[234] )
        initstate=`expr "$1" : '-i)'`
        echo "$0:  Initstate $i is not for system shutdown"
        exit 1
        ;;
    -y )
        askconfirmation=
        ;;
    -* )
        echo "Illegal flag argument '$1'"
        exit 1
        ;;
    * )
        echo "Usage: $0 [-y] [-g<grace>] [-i<initstate>]"
        exit 1
        esac
        shift
done
if [ -n "${askconfirmation}" -a -x /etc/ckbupscd ]
        then

#       Check to see if backups are scheduled at this time

        BUPS=`/etc/ckbupscd`
        if [ "$BUPS" != "" ]
        then
        echo "$BUPS"
        echo "Do you wish to abort this shutdown and return
        to command level to do these backups? [y, n]
              read YORN
              if [ "$YORN" = "y" -o "$YORN" = "Y" ]
                    then
                    exit 1
              fi
        fi
fi
if [ -z "${TZ}"  -a  -r /etc/TIMEZONE ]
then
   . /etc/TIMEZONE
fi
echo '
Shutdown started.    date
echo
sync
cd /
trap "exit 1"  1 2 15
a="`who  |  wc -l`"
if [ ${a} -gt 1  -a  ${grace} -gt 0 ]
then
su adm -c /etc/wall<<-!
        The system will be shut down in ${grace} seconds.
```

```
        Please log off now.
!
        sleep ${grace}
fi
/etc/wall <<-!
        THE SYSTEM IS BEING SHUT DOWN NOW ! ! !
        Log off now or risk your files being damaged.
!
sleep ${grace}
if [ ${askconfirmation} ]
then
        echo "Do you want to continue? (y or n):   \c"
        read b
else
        b=y
fi
if [ "$b" != "y" ]
then
        /etc/wall <<-\!
        False Alarm:  The system will not be brought down.
        !
        echo 'Shut down aborted.'
        exit 1
fi
case "${initstate}" in
s | S )
    . /etc/rc0
esac
/etc/init ${initstate}
```

The executable files in **/etc/shutdown.d** perform the various functions required during the transition to the single-user state (run levels 1, s, or S). This directory is only maintained for compatibility reasons.

SEE ALSO

shutdown(1), rc(4)

NAME

spellhist – unmatched spelling words file

DESCRIPTION

If the spell utility is installed, a history of all words that **spell**(1) fails to match is kept in the **/usr/lib/spell/spellhist** file. Periodically, this file should be reviewed for words that should be added to the dictionary. After the **spellhist** file is reviewed, it can be cleared.

NAME

sulog – log of substitute user events

SYNOPSIS

The **/usr/adm/sulog** file contains a history of substitute user (**su**) command usage. As a security measure, this file should not be readable by **others**. The */usr/adm/sulog* file should be periodically truncated to keep the size of the file within a reasonable limit. Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **sulog** file. To use one of these functions, add the appropriate command line to the */usr/spool/cron/crontab/root*, */etc/shutdown.d/...*, or */etc/rc.d/*, *rc2.d*, *rc3.d* ... file. The following command lines limit the size of the log file to the last 100 lines in the file:

```
tail -100 /usr/adm/sulog > /tmp/sulog
mv /tmp/sulog /usr/adm/sulog
```

EXAMPLES

The following is a typical **/usr/adm/sulog** file.

```
SU 08/18 12:35 + console root-sysadm
SU 08/18 16:11 + console root-sysadm
SU 08/18 16:16 + console root-sysadm
SU 08/18 23:45 + tty?? root-uucp
SU 08/19 11:53 + console root-sysadm
SU 08/19 15:25 + console root-sysadm
SU 08/19 23:45 + tty?? root-uucp
SU 08/20 10:16 + console root-adm
SU 08/20 10:33 + tty24 rar-root
SU 08/20 10:42 + console root-sysadm
SU 08/20 10:59 + console root-root
SU 08/20 11:01 + console root-sysadm
SU 08/20 12:36 + tty11 bin-bin
SU 08/20 12:37 + tty11 tws-bin
SU 08/20 14:42 - tty24 awa-sys
SU 08/20 14:47 - tty24 awa-sys
SU 08/20 14:48 + tty24 awa-root
SU 08/20 15:44 + console root-sysadm
```

SEE ALSO

su(1M)

NAME

syms – common object file symbol table format

SYNOPSIS

#include <syms.h>

DESCRIPTION

Common object files contain information to support symbolic software testing (debugging). Line number entries, **linenum**(4), and extensive symbolic information permit testing at the C **source** level. Every object file's symbol table is organized as shown below.

File name 1.
    Function 1.
        Local symbols for function 1.
    Function 2.
        Local symbols for function 2.
    ...
    Static externs for file 1.

File name 2.
    Function 1.
        Local symbols for function 1.
    Function 2.
        Local symbols for function 2.
    ...
    Static externs for file 2.
...

Defined global symbols.
Undefined global symbols.

The entry for a symbol is a fixed-length structure. The members of the structure hold the name (null padded), its value, and other information. The C structure is as follows:

```
#define  SYMNMLEN     8
#define  FILNMLEN     14
#define  DIMNUM       4

struct  syment
{
    union                           /* all ways to get symbol name */
    {
        char        _n_name[SYMNMLEN]; /* symbol name */
        struct
        {
            long    _n_zeros;       /* == 0L when in string table */
            long    _n_offset;      /* location of name in table */
        } _n_n;
        char        *_n_nptr[2];    /* allows overlaying */
    } _n;
    long            n_value;        /* value of symbol */
    short           n_scnum;        /* section number */
```

```
    unsigned short  n_type;         /* type and derived type */
    char            n_sclass;       /* storage class */
    char            n_numaux;       /* number of aux entries */
};

#define  n_name    _n._n_name
#define  n_zeros   _n._n_n._n_zeros
#define  n_offset  _n._n_n._n_offset
#define  n_nptr    _n._n_nptr[1]
```

Meaningful values and explanations for them are given in both **<syms.h>** and *Common Object File Format*(COFF). Refer to these sources for more information.

Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent
{
      struct
      {
            long          x_tagndx;
            union
            {
                  struct
                  {
                        unsigned shortx_lnno;
                        unsigned shortx_size;
                  } x_lnsz;
                  long    x_fsize;
            } x_misc;
            union
            {
                  struct
                  {
                        long    x_lnnoptr;
                        long    x_endndx;
                  }       x_fcn;
                  struct
                  {
                        unsigned shortx_dimen[DIMNUM];
                  }       x_ary;
            }             x_fcnary;
            unsigned short  x_tvndx;
      }     x_sym;
      struct
      {
            char    x_fname[FILNMLEN];
      }     x_file;
        struct
        {
              long     x_scnlen;
```

```
                unsigned short  x_nreloc;
                unsigned short  x_nlinno;
        }       x_scn;

    struct
    {
        long            x_tvfill;
        unsigned short      x_tvlen;
        unsigned short      x_tvran[2];
    }   x_tv;
};
```

Indexes of symbol table entries begin at **zero.**

SEE ALSO

sdb(1), a.out(4), linenum(4)

WARNINGS

On machines on which **int**s are equivalent to **long**s, all **long**s have their type changed to **int.** Thus the information about which symbols are declared as **long**s and which as **int**s does not show up in the symbol table.

NAME

sys_config – format of a system configuration file

DESCRIPTION

The **sys_config** file describes how the kernel is to be built and what devices are to be available when the kernel boots. The directives below may appear in any order. Options may appear as a single quantity prepended with an equal sign, as a comma-separated list in parentheses, or as a line-delimited list enclosed in braces. Comments are preceded with a pound (#) sign.

The **sys_config** files must contain at least one of the following sections:

arch_type
version
modules
device_spec

The system configuration files usually contain the following sections:

makeopts
conf
confh

The directives are as follows:

Directive	**Description**
import { *options* }	The named product system configuration files must be present, and parsed before this one
checkfor { *options* }	The named product system configuration files if present, must be parsed before this one
arch_type { *option* }	Architecture type - SB8000
version { *options* }	Version support
description { "*option*" }	Module description
environment { *option* }	Anticipated usage environment
archive { *option* }	Product archive
makefile { *option* }	Name of skeleton makefile
modules { *options* }	List of compile time modules
if*TYPE* (*parameter*) {	Conditional if*TYPE* (e.g. **ifdef**) TYPE is one of: **config, nconfig, def, ndef, product, nproduct.**
{ statements }	- *true clause*
} **elsif***TYPE* (*parameter*) {	Optional else if*TYPE* (e.g. **elsifdef**) TYPE is one of: **config, nconfig, def, ndef, product, nproduct.**
{ statements }	- *true clause*
} **else** {	Optional else
{ statements }	- *false clause*
}	

makeopts { *options* }	Makefile options
conf { *options* }	*conf.c* options
confh { *options* }	*config.h* options
%RULES { *rstring* } { *string* }	Make rules
{ dev } **on** { *type* } **unit** { number }	Device specification
pseudo_device { *p_spec* }	Pseudo-device specification
inits { *options* }	Declare these initialization functions
print { *option* }	Print argument
fatal { *option* }	Print argument and abort configuration
trace	Trace toggle
include { *option* }	Include the named file
declare [*level*] { *params* }	Declare configurable parameters
default { *defaults* }	Define default parameter values
new_default { *defaults* }	Define new default parameter values
adjust { *param_exprs* }	Adjust parameter values
set { *param_exprs* }	Set parameter values
bound { *bound_exprs* }	Bound parameter values
assert { *assertions* }	Assert parameter values

The directive modifiers are as follows:

Modifier	**Description**
level	Defines parameter visibility level [1..10]
params	List of parameters in the form: *name type* { *"description"* }
defaults	List of default parameter values: *name* { *expr* } *name* { *expr* ; *bound* }
param_exprs	List of expressions adjusting or setting parameter values: *name* { *expr* }
bound_exprs	List of parameter bound expressions: *name* { *bound* }
assertions	List of parameter value assertions: { *expr relop expr* ; *"assert text"* }
bound	Bound interval for parameter: *low_bound_expr* .. .. *high_bound_expr* *low_bound_expr* .. *high_bound_expr*
expr	A simple arithmetic expression
relop	A relational operator: **<= < = > >=**

SEE ALSO

config (1m), mod_config (4)

NAME

syskeymap – identify keys to internal menu system functions

ptx/ADMIN FEATURE CONTROL FLAG RECORDS (FLAG)

The FLAG records enable and disable menu system functions. FLAG records affect the way the menus and forms are displayed and how the user interacts with the programs.

Record Identifier

This field has the constant value of **flag** to identify the record type.

ptx/ADMIN Feature

This field selects the features that can be be enabled/disabled by the next field.

DUMBTERM	Is this a dumb terminal?
UNDINPUT	Display input fields with underline video attribute?
REVTITLE	Display title in reverse video attribute?
REVEMSG	Display error messages in reverse video attribute?
ATTRTRIM	Display menu trim with selected video attribute?
BOLDGRANT	Display authorized menu options in bold video attribute?
DIMDENY	Display unauthorized menu options in dim video attribute?
BLINKRUN	Display executing menu options in blinking video attribute?
REVMBAR	Display menu bar in reverse video attribute?

Feature On/Off

If this field is **YES**, the feature will be enabled. If this field is **NO**, the feature will be disabled.

ptx/ADMIN FUNCTION KEY ASSIGNMENT RECORDS (KEYMAP)

The KEYMAP records are used to rename or reassign keys to internal menu system functions. This feature allows the System Administrator to change the control sequence and/or key label recognized by the menu or forms programs. For example, terminals that have a non-standard layout or alternate keycaps may have functions mapped to new keys. The descriptions of the functions will appear on HELP displays, and the bottom line of menus and forms will reflect the new key and/or labels.

Record Identifier

This field has the constant value of **KEYMAP** to identify the record type.

ptx/ADMIN Feature

This field selects the function that is remapped by the next field.

BACK SPACE	destructive backspace
CANCEL	cancel
CHDIR	change directory
CLRFIELD	clear field

COPY	copy
DELCHAR	delete character
DOWNARROW	down arrow
EXIT	exit
EXTEND	extend field
HELP	help command
HOME	home
INSCHAR	insert character
KEYHELP	keyboard help
LEFTARROW	left arrow (non-destructive backspace)
LOCK	lock menu
MSGLINE	message line
PREVIOUS	previous
PROMPT	prompt
REFRESH	display refresh
RIGHTARROW	right arrow
ROLLDOWN	roll down
ROLLUP	roll up
RUN	run/send/execute
SHELLCMD	shell command
SPECIAL1	special function 1
SPECIAL2	special function 2
SPECIAL3	special function 3
SPECIAL4	special function 4
SPECIAL5	special function 5
SPECIAL6	special function 6
SPECIAL7	special function 7
SPECIAL8	special function 8
SPECIAL9	special function 9
SPECIAL10	special function 10
UPARROW	up arrow

Alternate Label for Function
This field allows you to change the external name of the menu system function. For example, if you want to change the description of the Page Down function to Roll Down on menus and forms, enter **Page Down** in this field.

Alternate Mapping(s) for Function
This field consists of two subfields separated by a comma. The first field is the substitute label for the key. The second field is the substitue ASCII Sequence

or Terminfo key identifier associated with the function.

To map a function to a function key, enter the number of the function key followed by the label on the key cap. Function key numbers are listed below:

Function Key	**Number**
Down Arrow	**402**
Up Arrow	**403**
Left Arrow	**404**
Right Arrow	**405**
Home Key	**406**
Backspace Key	**407**
Function Keys	**410+***function key number*
Delete Character	**512**
Insert Character	**513**
Clear Field	**515**
Next Page	**522**
Previous Page	**523**
Send/Execute	**527**

To substitute an ASCII sequence, enter the octal value of the ASCII character followed by a description of the sequence. Do not substitute printable characters (for example, letters or numbers) because the results are unpredictable.

FILES

${TERMINFO}/*/	*${TERM}* Terminal Description (terminfo)
${TFADMFILE}/keymaps/	*${TERM}* Terminal Specific Alternate Keymap

NAME

task – global security table

DESCRIPTION

The global task table is used as a central repository that contains security and implementation information for menu system programs. The information in each security entry can be accessed by menu system menus and forms. A menu accesses the task table to find out if the user is authorized to execute the task, and to find out how to execute the task. A form may access the task table to see if the user has authority to a menu or a form.

FIELDS

Task Name

The name of the task. The name must be unique and from 1 to 14 characters in length. The task name is upper/lower case sensitive. This is the key to the record and all extractions by the menu and menu system commands are based on this value. This is a required field.

Task Type

The task type option tells the menu program if the task is a menu, program, or option modifier. The valid task type codes are:

M Menu
P Program
O Option modifier

This is a required field.

Task Description

This describes the option. It is displayed by the *dspsecent* and *dsptsk* commands and is for reference only. This is an optional field.

Execution Type

Use this option to identify how menu driver executes the program task types. For task types other than program, this field is ignored. The valid execution type codes are listed below:

SHELL Shell required to execute task. (Default)
NO SHELL No shell is required to execute task.
EDC Menu system required to execute task.

This is an optional field.

Clear Screen

This option controls if the screen is cleared before the program is executed and if the screen is refreshed (redisplayed) after the program's completion. For task types other than program, this field is ignored. The valid clear screen codes are listed below:

BEHIND Do not clear or redisplay the menu.
CLEAR Clear menu before task and redisplay after.
REFRESH Do not clear menu, but redisplay menu after task completion.

This is an optional field.

Hold Screen

This option controls the screen after a program is executed. When a program task type uses the Hold Screen option, the user must press the RETURN key

to refresh the menu. This option is used when a command produces output to the screen. It "freezes" the output on the screen and allows the user to review the data.

Use this option with the execution type of program (P) and a clear screen code that refreshs the menu. For task types other than program, this field is ignored.

The valid hold screen codes are listed below:

NOHOLD	Do not hold screen after program execution.
HOLD	Hold screen after program execution.

This is an optional field.

Authorized Group List

This option controls the groups that can access the task. If there is more than one group, separate the names with commas, or separate the names with spaces and enclose the list in quotes. Each group name is verified against the group file. The group list option is optional.

Authorized User List

This option controls the users that can access the task. If there is more than one user, separate the names with commas, or separate the names with spaces and enclose the list in quotes. Each user name is verified against the system password file. To allow all users access to the task, enter **PUBLIC.** This is an optional field.

Command String/Menu Name

The use of the command option is dependent on the execution type. The following command strings apply to task types:

P	Program or command
M	Menu
O	Option modifier

This is a required field.

Help Command

The menu program uses the help command. If there is a value in this field, the value is is concatenated with the value in the next field (the help file) and then executed. If the value is HELPCMD, the system control file is opened and the default Help Command (such as 'man') is substituted. This is an optional field.

Help File

The help file is used by the menu program. If there is a value in the previous field, Help Command, the value of this field is concatenated with the help command and then executed. If the help command field is null, the file is opened by the menu driver and the text delimited by the Help Keywords (the next field) is displayed in the help window. This is an optional field.

Help Keywords

The help keywords are used by the menu program. If help text exists (as determined by the Help Command and Help File), the value in this field is used to delimit the text relevant to the current option. If this field is null, all text is passed through to the help window. If one or more words (comma separated) are specified, lines starting with a period (in the first column only)

followed directly by a keyword *(.keyword)* will initiate extraction of the text. Lines starting with a period and a tilde *(.~keyword)* will terminate extraction of the text. The entire file is scanned for all keywords that are noted. This is an optional field.

SEE ALSO

menu(1M)

NOTES

Only the superuser (root) can update this file. All users can read this file.

NAME

term – format of compiled term file

SYNOPSIS

/usr/lib/terminfo/?/*

DESCRIPTION

Compiled **terminfo**(4) descriptions are placed under the directory */usr/lib/terminfo*. In order to avoid a linear search of a huge operating system directory, a two-level scheme is used: */usr/lib/terminfo/c/name* where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, `att4425` can be found in the file */usr/lib/terminfo/a/att4425*. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it is the same on all hardware. An 8-bit byte is assumed, but no assumptions about byte ordering or sign extension are made. Thus, these binary **terminfo**(4) files can be transported to other hardware with 8-bit bytes.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value **–1** is represented by **0377,0377**, and the value **–2** is represented by **0376,0377**; other negative values are illegal. Computers where this does not correspond to the hardware read the integers as two bytes and compute the result, making the compiled entries portable between machine types. The **–1** generally means that a capability is missing from this terminal. The **–2** means that the capability has been canceled in the **terminfo**(4) source and also is to be considered missing.

The compiled file is created from the source file descriptions of the terminals (see the **–I** option of **infocmp**(1M)) by using the **terminfo**(4) compiler, **tic**(1M), and read by the routine **setupterm**( ). (See **curses**(3X).) The file is divided into six parts: the header, terminal names, boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are (1) the magic number (octal **0432**); (2) the size, in bytes, of the names section; (3) the number of bytes in the boolean section; (4) the number of short integers in the numbers section; (5) the number of offsets (short integers) in the strings section; (6) the size, in bytes, of the string table.

The terminal names section comes next. It contains the first line of the **terminfo**(4) description, listing the various names for the terminal, separated by the bar (|) character (see **term**(5)). The section is terminated with an ASCII NULL character.

The boolean flags have one byte for each flag. This byte is either **0** or **1** as the flag is present or absent. The value of **2** means that the flag has been canceled. The capabilities are in the same order as the file *<term.h>*.

Between the boolean section and the number section, a null byte is inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short word boundary.

The numbers section is similar to the boolean flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is **–1** or **–2**, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of **–1** or **–2** means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^X or \c notation are stored in their interpreted form, not the printing representation. Padding information ($<nn>) and parameter information (%x) are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null terminated.

Note that it is possible for **setupterm**( ) to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since **setupterm**( ) has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine **setupterm**( ) must be prepared for both possibilities – this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

As an example, an octal dump of the description for the AT&T Model 37 KSR is included:

```
37|tty37|AT&T model 37 teletype,
    hc, os, xon,
    bel=^G, cr=\r, cubl=\b, cudl=\n, cuul=\E7, hd=\E9,
    hu=\E8, ind=\n,

0000000 032 001    \0 032  \0 013  \0 021 001   3  \0   3   7   |   t
0000020   t   y   3   7   |   A   T   &   T     m   o   d   e   l
0000040   3   7     t   e   l   e   t   y   p   e  \0  \0  \0  \0  \0
0000060 \0  \0  \0 001  \0  \0  \0  \0  \0  \0  \0 001  \0  \0  \0  \0
0000100 001 \0  \0  \0  \0  \0 377 377 377 377 377 377 377 377 377 377
0000120 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377   &  \0
0000140    \0 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000160 377 377   "  \0 377 377 377 377   (  \0 377 377 377 377 377 377
0000200 377 377   0  \0 377 377 377 377 377 377 377 377   -  \0 377 377
0000220 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000520 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377   $  \0
0000540 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377   *  \0
0000560 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*                                                               '
0001160 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377   3   7
0001200   |   t   t   y   3   7   |   A   T   &   T     m   o   d   e
0001220   l     3   7       t   e   l   e   t   y   p   e  \0  \r  \0
0001240  \n  \0  \n  \0 007  \0  \b  \0 033   8  \0 033   9  \0 033   7
0001260  \0  \0
0001261
```

Some limitations: total compiled entries cannot exceed 4096 bytes; all entries in the name field cannot exceed 128 bytes.

FILES

/usr/lib/terminfo/?/*	Compiled terminal description database
/usr/include/term.h	**terminfo**(4) header file

SEE ALSO

infocmp(1M), curses(3X), terminfo(4), term(5)

NAME

termcap – terminal capability data base

SYNOPSIS

/etc/termcap

DESCRIPTION

termcap is a database file that describes terminal capabilities. The *termcap* database is obsolete and is provided for backward compatibility; the *terminfo*(4) database should be used instead. Entries in *termcap* describe terminal capabilities and how operations are performed. Padding requirements and initialization sequences are included in *termcap.*

Entries in *termcap* consist of a number of colon (:) separated fields. The first entry for each terminal gives the name or names that are known for the terminal, separated by pipe (|) characters. The first name is always 2 characters long and is used by older Version 6 UNIX systems which store the terminal type in a 16 bit word in a systemwide data base. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks; the last name may well contain blanks for readability.

CAPABILITIES

(P) indicates padding may be specified
(P*) indicates that padding may be based on no. lines affected

Name	Type	Pad?	Description
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not ^H
bs	bool		Terminal can backspace with ^H
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
ch	str	(P)	Like cm but horizontal motion only, line stays same
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
cr	str	(P*)	Carriage return, (default ^M)
cs	str	(P)	Change scrolling region (vt100), like cm
cv	str	(P)	Like ch but vertical only.
da	bool		Display may be retained above
dB	num		Number of millisec of bs delay needed
db	bool		Display may be retained below
dC	num		Number of millisec of cr delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)

dN	num		Number of millisec of nl delay needed
do	str		Down one line
ds	str		Display status line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give :ei=: if **ic**
eo	str		Can erase overstrikes with a blank
es	bool		Supports escape sequences on status line
ff	str	(P*)	Hardcopy terminal page eject (default ^L)
fs	str		End writing to status line
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
ho	str	(P)	Home cursor (if no **cm**)
hs	bool		Has a status line
hu	str		Half-line up (reverse 1/2 linefeed)
hz	bool		Hazeltine; can't print ~'s
i2	str		Secondary terminal initialization string
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	bool		Insert mode (enter); give :im=: if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by other function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of keypad transmit mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of other keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in keypad transmit mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on other function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map, used by vi version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor.
ms	bool		Safe to move while in standout and underline mode
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default \n)
ns	bool		Terminal is a CRT but doesn't scroll.
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
pt	bool		Has hardware tabs (may need to be set with **is**)
se	str		End stand out mode
sf	str	(P)	Scroll forwards

sg	num		Number of blank chars left by so or se
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use **cm**
ti	str		String to begin programs that use **cm**
ts	str		Begin writing to status line
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by us or ue
ul	bool		Terminal underlines though it doesn't overstrike
up	str		Upline (cursor up)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like ce \r \n (Delta Data)
xs	bool		Standout not erased by writing over it (HP 264?)
xt	bool		Tabs are destructive, magic so char (Teleray 1061)

A Sample Entry

The following entry, which describes the Concept–100, is among the more complex entries in the *termcap* file as of this writing. (This particular concept entry is outdated, and is used as an example only.)

```
c1|c100|concept100:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200:\
:al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:cm=\Ea%+ %+ :co#80:\
:dc=16\E^A:dl=3*\E^B:ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
:se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and that empty fields may be included for readability (here between the last field on a line and the first field on the next). Capabilities in *termcap* are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two letter codes. For instance, the fact that the Concept has automatic margins (i.e. an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **co** which indicates the number of columns the terminal has gives the value '80' for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the

remainder of the string is sent to provide this delay. The delay can be either a integer, e.g. '20', or an integer followed by an '*', i.e. '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \r \t \b \f** give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a ****, and the characters **^** and **** may be given as **\^** and ****. If it is necessary to place a **:** in a capability it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability it must be encoded as **\200**. The routines which deal with *termcap* use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *termcap* and to build up a description gradually, using partial descriptions with *ex* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *termcap* file to describe it or bugs in *ex*. To easily test a new terminal description you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in */etc/termcap*. TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor. (This only works on version 7 systems.)

Basic capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, then this is given by the **cl** string capability. If the terminal can backspace, then it should have the **bs** capability, unless a backspace is accomplished by a character other than **^H** (ugh) in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability.

A very important point here is that the local cursor motions encoded in *termcap* are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the *termcap* file usually assumes that this is on (i.e., **am**).

These capabilities suffice to describe hardcopy and glass-tty terminals. Thus the model 33 teletype is described as

t3 | 33 | tty33:co#72:os

while the Lear Siegler ADM-3 is described as

cl | adm3 | 3 | lsi adm3:am:bs:cl=^Z:li#24:co#80

Cursor addressing

Cursor addressing in the terminal is described by a **cm** string capability, with *printf*(3S) like escapes **%x** in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, then its arguments are the line and then the column to which motion is desired, and the **%** encodings have the following meanings:

%d	as in *printf*, 0 origin
%2	like %2d
%3	like %3d
%.	like %c
%+x	adds *x* to value, then %.
%>xy	if value > x adds y, no output.
%r	reverses order of line and column, no output
%i	increments line/column (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output.
%D	Reverse coding (x-2*(x%16)), no output. (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is cm=6\E&%r%2c%2Y. The Microterm ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, cm=^T%.%.. Terminals which use %. need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t**, **\n** ^**D** and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus cm=\E=%+ %+ .

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, this should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen) then this can be given as **ho**; similarly a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

Area clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **ce**. If the terminal can clear from the current position to the end of the display, then this should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **al**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl**; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as **sb**, but just **al** suffices. If the terminal can retain display memory above then the **da** capability should be given; if display memory can be retained below then **db** should be given. These let the editor understand that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using *termcap*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type abc def using local cursor motions (not spaces) between the abc and the def. Then position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the abc shifts over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for insert null. If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines. We have seen no terminals which have an insert mode not not falling into one of these two classes.

The editor can handle both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give as **im** the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Give as **ei** the sequence to leave insert mode (give this, with an empty value also if you gave **im** so). Now give as **ic** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ic**, terminals which send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give

this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g. if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode these can be given as **so** and **se** respectively. If there are several flavors of standout mode (such as inverse video, blinking, or underlining – half bright is not usually an acceptable standout mode unless the terminal is in inverse video mode constantly) the preferred mode is inverse video by itself. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **ug** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **us** and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

Many terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of *ex*, this can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change, e.g., from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl, kr, ku, kd,** and **kh** respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0, l1, ..., l9**. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the *termcap* 2 letter codes can be given in the **ko** capability, for example, :ko=cl,ll,sf,sb:, which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete but still in use in version 2 of vi, which must be run on some minicomputers due to memory limitations. This field is redundant with **kl, kr, ku, kd,** and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, the mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than ^I to tab, then this can be given as **ta**.

Hazeltine terminals, which don't allow tilde characters to be printed should indicate **hz**. Datamedia terminals, which echo carriage-return linefeed for carriage return and then ignore a following linefeed should indicate **nc**. Early Concept terminals, which ignore a linefeed immediately after an **am** wrap, should indicate **xn**. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt**. Other specific terminal problems may be corrected by adding more capabilities of the form **x*x***.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** will be printed before **if**. This is useful where **if** is */usr/lib/tabset/std* but **is** clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be *last* and the combined length of the two entries must not exceed 1024. Since *termlib* routines

search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be canceled with **xx@** where xx is the capability. For example, the entry

hn | 2621nl:ks@:ke@:tc=2621:

defines a 2621nl that does not have the **ks** or **ke** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

/etc/termcap file containing terminal descriptions

SEE ALSO

terminfo(4), captoinfo(1M)

AUTHOR

William Joy
Mark Horton added underlining and keypad support

BUGS

The **ma**, **vs**, and **ve** entries are specific to the *vi* program.

Not all programs support all entries. There are entries that are not supported by any program.

NAME

terminfo – terminal capability database

SYNOPSIS

/usr/lib/terminfo/?/*

DESCRIPTION

terminfo is a database, produced by **tic**(1M), that describes the capabilities of devices such as terminals and printers. Devices are described in *terminfo* source files by specifying a set of capabilities, by quantifying certain aspects of the device, and by specifying character sequences that effect particular results. This database is often used by screen oriented applications such as **vi**(1) and **curses**(3X), as well as by some operating system commands such as **ls**(1) and **pg**(1). This usage allows them to work with a variety of devices without changes to the programs. To obtain the source description for a device, use the **infocmp**(1M) command.

terminfo source files consist of one or more device description entries. Each entry consists of a header (beginning in column 1) and one or more lines that list the features for that particular device. Each line in a *terminfo* source file, except the header, must be indented with one or more white spaces (either spaces or tabs); and each line must end in a comma (,).

Entries consist of a number of comma-separated fields; white space after each comma is ignored. Embedded commas must be escaped by using a backslash. The following example shows the format of a *terminfo* source file:

```
alias1 | alias2 | ... | aliasN | longname,
 #24,
 home=\Eeh,
```

The first line, commonly referred to as the header line, must begin in column one and must contain at least two aliases, separated by vertical bars. The last field in the header line must be the long name of the device and it may contain any string. Alias names must be unique in the *terminfo* database and they must conform to operating system file naming conventions (see **tic**(1M)); for example they cannot contain white space or slashes.

Every device must be assigned a name, such as "att5425" (for the AT&T model 5425 device). Device names (except the long name) should be should not contain hyphens because hyphens are reserved for use when adding suffixes that indicate special modes for hardware. or modes for user preference.

To assign a special modes to a particular device, append a suffix, consisting of a hyphen and an indicator of the mode, to the device name. For example, the *-w* suffix means *wide mode*; when specified, it allows for a width of 132 columns instead of the standard 80 columns. Therefore, if you want to use an AT&T 5425 device set to wide mode, name the device "att5410-w." Use the following suffixes where possible.

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	5410-w

-am	With auto. margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
-*n*	Number of lines on the screen	2300-40
-na	No arrow keys (leave them in local)	c100-na
-*n*p	Number of pages of memory	c100-4p
-rv	Reverse video	4415-rv

PART 1 DEVICE CAPABILITIES

Three types of capabilities are included in the *terminfo* file:

- Boolean capabilities which show that a device has or does not have a particular feature.
- Numeric capabilities which quantify particular features of a device.
- String capabilities which provide sequences that can be used to perform particular operations on devices.

In the following tables, a **Variable** is the name by which a C programmer accesses a capability (at the *terminfo* level). A **Capname** is the short name for a capability specified in the *terminfo* source file. It is used by a person updating the source file and by the **tput**(1) command. A **Termcap Code** is a two-letter sequence that corresponds to the *termcap* capability name. (Note that *termcap* is no longer supported.)

Capability names have an informal limit of five characters to keep them short. Whenever possible, capability names are chosen to be the same as or similar to those specified by the ANSI X3.64-1979 standard. Semantics are also intended to match those of the ANSI standard.

All string capabilities listed below may have padding specified, with the exception of those used for input. Input capabilities, listed under the **Strings** section in the following tables, have names beginning with **key_**. The *#i* symbol in the description field of the following tables refers to the *i*th parameter.

Booleans

Variable	Cap-name	Termcap Code	Description
auto_left_margin	bw	bw	**cubl** wraps from column 0 to last column
auto_right_margin	am	am	Terminal has automatic margins
back_color_erase	bce	be	Screen erased with background color
can_change	ccc	cc	Terminal can re-define existing color
ceol_standout_glitch	xhp	xs	Standout not erased by overwriting (hp)
col_addr_glitch	xhpa	YA	Only positive motion for **hpa/mhpa** caps
cpi_changes_res	cpix	YF	Changing char pitch changes resolution
cr_cancels_micro_mode	crxm	YB	Using **cr** turns off micro mode
eat_newline_glitch	xenl	xn	Newline ignored after 80 columns (*Concept*)
erase_overstrike	eo	eo	Can erase overstrikes with a blank
generic_type	gn	gn	Generic line type (e.g., dialup, switch)
hard_copy	hc	hc	Hardcopy terminal

Variable	Cap-name	Termcap Code	Description
hard_cursor	chts	HC	Cursor is hard to see
has_meta_key	km	km	Has a meta key (shift, sets parity bit)
has_print_wheel	daisy	YC	Printer needs operator to change char set
has_status_line	hs	hs	Has extra "status line"
hue_lightness_saturation	hls	hl	Terminal uses only HLS color notation (Tektronix)
insert_null_glitch	in	in	Insert mode distinguishes nulls
lpi_changes_res	lpix	YG	Changing line pitch changes resolution
memory_above	da	da	Display may be retained above the screen
memory_below	db	db	Display may be retained below the screen
move_insert_mode	mir	mi	Safe to move while in insert mode
move_standout_mode	msgr	ms	Safe to move in standout modes
needs_xon_xoff	nxon	nx	Padding won't work, xon/xoff required
no_esc_ctlc	xsb	xb	Beehive (f1=escape, f2=ctrl C)
non_rev_rmcup	nrrmc	NR	**smcup** does not reverse **rmcup**
no_pad_char	npc	NP	Pad character doesn't exist
over_strike	os	os	Terminal overstrikes on hard-copy terminal
prtr_silent	mc5i	5i	Printer won't echo on screen
row_addr_glitch	xvpa	YD	Only positive motion for **vpa/mvpa** caps
semi_auto_right_margin	sam	YE	Printing in last column causes **cr**
status_line_esc_ok	eslok	es	Escape can be used on the status line
dest_tabs_magic_smso	xt	xt	Destructive tabs, magic **smso** char (t1061)
tilde_glitch	hz	hz	Hazeltine; can't print tilde (~)
transparent_underline	ul	ul	Underline character overstrikes
xon_xoff	xon	xo	Terminal uses xon/xoff handshaking

Numbers

Variable	Cap-name	Termcap Code	Description
buffer_capacity	bufsz	Ya	Number of bytes buffered before printing
columns	cols	co	Number of columns in a line
dot_vert_spacing	spinv	Yb	Spacing of pins vertically in pins per inch
dot_horz_spacing	spinh	Yc	Spacing of dots horizontally in dots per inch
init_tabs	it	it	Tabs initially every # spaces
label_height	lh	lh	Number of rows in each label
label_width	lw	lw	Number of columns in each label
lines	lines	li	Number of lines on a screen or a page
lines_of_memory	lm	lm	Lines of memory if > **lines**; 0 means varies
magic_cookie_glitch	xmc	sg	Number of blank characters left by **smso** or **rmso**
max_colors	colors	Co	Maximum number of colors on the screen
max_micro_address	maddr	Yd	Maximum value in **micro_..._address**
max_micro_jump	mjump	Ye	Maximum value in **parm_..._micro**
max_pairs	pairs	pa	Maximum number of color-pairs on the screen
micro_col_size	mcs	Yf	Character step size when in micro mode
micro_line_size	mls	Yg	Line step size when in micro mode
no_color_video	ncv	NC	Video attributes that can't be used with colors

Variable	Cap-name	Termcap Code	Description
number_of_pins	npins	Yh	Number of pins in print-head
num_labels	nlab	Nl	Number of labels on screen (start at 1)
output_res_char	orc	Yi	Horizontal resolution in units per character
output_res_line	orl	Yj	Vertical resolution in units per line
output_res_horz_inch	orhi	Yk	Horizontal resolution in units per inch
output_res_vert_inch	orvi	Yl	Vertical resolution in units per inch
padding_baud_rate	pb	pb	Lowest baud rate where padding needed
virtual_terminal	vt	vt	Virtual terminal number (DYNIX/ptx)
wide_char_size	widcs	Yn	Char step size when in double wide mode
width_status_line	wsl	ws	Number of columns in status line

Strings

Variable	Cap-name	Termcap Code	Description
acs_chars	acsc	ac	Graphic charset pairs aAbBcC - def=vt100
back_tab	cbt	bt	Back tab
bell	bel	bl	Audible signal (bell)
carriage_return	cr	cr	Carriage return
change_char_pitch	cpi	ZA	Change number of characters per inch
change_line_pitch	lpi	ZB	Change number of lines per inch
change_res_horz	chr	ZC	Change horizontal resolution
change_res_vert	cvr	ZD	Change vertical resolution
change_scroll_region	csr	cs	Change to lines #1 through #2 (vt100)
char_padding	rmp	rP	Like ip but when in replace mode
char_set_names	csnm	Zy	List of character set names
clear_all_tabs	tbc	ct	Clear all tab stops
clear_margins	mgc	MC	Clear all margins (top, bottom, and sides)
clear_screen	clear	cl	Clear screen and home cursor
clr_bol	el1	cb	Clear to beginning of line, inclusive
clr_eol	el	ce	Clear to end of line
clr_eos	ed	cd	Clear to end of display
column_address	hpa	ch	Horizontal position absolute
command_character	cmdch	CC	Terminal settable cmd character in prototype
cursor_address	cup	cm	Move to row #1 col #2
cursor_down	cud1	do	Down one line
cursor_home	home	ho	Home cursor (if no cup)
cursor_invisible	civis	vi	Make cursor invisible
cursor_left	cub1	le	Move left one space.
cursor_mem_address	mrcup	CM	Memory relative cursor addressing
cursor_normal	cnorm	ve	Make cursor appear normal (undo vs/vi)
cursor_right	cuf1	nd	Non-destruct space (cursor or carriage right)
cursor_to_ll	ll	ll	Last line, first column (if no cup)
cursor_up	cuu1	up	Upline (cursor up)
cursor_visible	cvvis	vs	Make cursor very visible
define_char	defc	ZE	Define a character in a character set †

Variable	Cap-name	Termcap Code	Description
delete_character	dch1	dc	Delete character
delete_line	dl1	dl	Delete line
dis_status_line	dsl	ds	Disable status line
down_half_line	hd	hd	Half-line down (forward 1/2 linefeed)
ena_acs	enacs	eA	Enable alternate character set
enter_alt_charset_mode	smacs	as	Start alternate character set
enter_am_mode	smam	SA	Turn on automatic margins
enter_blink_mode	blink	mb	Turn on blinking
enter_bold_mode	bold	md	Turn on bold (extra bright) mode
enter_ca_mode	smcup	ti	String to begin programs that use **cup**
enter_delete_mode	smdc	dm	Delete mode (enter)
enter_dim_mode	dim	mh	Turn on half-bright mode
enter_doublewide_mode	swidm	ZF	Enable double wide printing
enter_draft_quality	sdrfq	ZG	Set draft quality print
enter_insert_mode	smir	im	Insert mode (enter)
enter_italics_mode	sitm	ZH	Enable italics
enter_leftward_mode	slm	ZI	Enable leftward carriage motion
enter_micro_mode	smicm	ZJ	Enable micro motion capabilities
enter_near_letter_quality	snlq	ZK	Set near-letter quality print
enter_normal_quality	snrmq	ZL	Set normal quality print
enter_protected_mode	prot	mp	Turn on protected mode
enter_reverse_mode	rev	mr	Turn on reverse video mode
enter_secure_mode	invis	mk	Turn on blank mode (chars invisible)
enter_shadow_mode	sshm	ZM	Enable shadow printing
enter_standout_mode	smso	so	Begin standout mode
enter_subscript_mode	ssubm	ZN	Enable subscript printing
enter_superscript_mode	ssupm	ZO	Enable superscript printing
enter_underline_mode	smul	us	Start underscore mode
enter_upward_mode	sum	ZP	Enable upward carriage motion
enter_xon_mode	smxon	SX	Turn on xon/xoff handshaking
erase_chars	ech	ec	Erase #1 characters
exit_alt_charset_mode	rmacs	ae	End alternate character set
exit_am_mode	rmam	RA	Turn off automatic margins
exit_attribute_mode	sgr0	me	Turn off all attributes
exit_ca_mode	rmcup	te	String to end programs that use **cup**
exit_delete_mode	rmdc	ed	End delete mode
exit_doublewide_mode	rwidm	ZQ	Disable double wide printing
exit_insert_mode	rmir	ei	End insert mode
exit_italics_mode	ritm	ZR	Disable italics
exit_leftward_mode	rlm	ZS	Enable right (normal) carriage motion
exit_micro_mode	rmicm	ZT	Disable micro motion capabilities
exit_shadow_mode	rshm	ZU	Disable shadow printing
exit_standout_mode	rmso	se	End standout mode
exit_subscript_mode	rsubm	ZV	Disable subscript printing

Variable	Cap-name	Termcap Code	Description
exit_superscript_mode	rsupm	ZW	Disable superscript printing
exit_underline_mode	rmul	ue	End underscore mode
exit_upward_mode	rum	ZX	Enable down (normal) carriage motion
exit_xon_mode	rmxon	RX	Turn off xon/xoff handshaking
flash_screen	flash	vb	Visible bell (may not move cursor)
form_feed	ff	ff	Hardcopy terminal page eject
from_status_line	fsl	fs	Return from status line
init_1string	is1	i1	Terminal or printer initialization string
init_2string	is2	is	Terminal or printer initialization string
init_3string	is3	i3	Terminal or printer initialization string
init_file	if	if	Name of initialization file
init_prog	iprog	iP	Path name of program for initialization
initialize_color	initc	Ic	Initialize the definition of color
initialize_pair	initp	Ip	Initialize color-pair
insert_character	ich1	ic	Insert character
insert_line	il1	al	Add new blank line
insert_padding	ip	ip	Insert pad after character inserted
key_a1	ka1	K1	KEY_A1, 0534, upper left of keypad
key_a3	ka3	K3	KEY_A3, 0535, upper right of keypad
key_b2	kb2	K2	KEY_B2, 0536, center of keypad
key_backspace	kbs	kb	KEY_BACKSPACE, 0407, by backspace key
key_beg	kbeg	@1	KEY_BEG, 0542, sent by beg(inning) key
key_btab	kcbt	kB	KEY_BTAB, 0541, sent by back-tab key
key_c1	kc1	K4	KEY_C1, 0537, lower left of keypad
key_c3	kc3	K5	KEY_C3, 0540, lower right of keypad
key_cancel	kcan	@2	KEY_CANCEL, 0543, sent by cancel key
key_catab	ktbc	ka	KEY_CATAB, 0526, sent by clear-all-tabs key
key_clear	kclr	kC	KEY_CLEAR, 0515, by cl-screen or erase key
key_close	kclo	@3	KEY_CLOSE, 0544, sent by close key
key_command	kcmd	@4	KEY_COMMAND, 0545, sent by cmd key
key_copy	kcpy	@5	KEY_COPY, 0546, sent by copy key
key_create	kcrt	@6	KEY_CREATE, 0547, sent by create key
key_ctab	kctab	kt	KEY_CTAB, 0525, sent by clear-tab key
key_dc	kdch1	kD	KEY_DC, 0512, sent by delete-character key
key_dl	kdl1	kL	KEY_DL, 0510, sent by delete-line key
key_down	kcud1	kd	KEY_DOWN, 0402, sent by down-arrow key
key_eic	krmir	kM	KEY_EIC, 0514, **rmir** or **smir** in insert mode
key_end	kend	@7	KEY_END, 0550, sent by end key
key_enter	kent	@8	KEY_ENTER, 0527, sent by enter/send key
key_eol	kel	kE	KEY_EOL, 0517, by clear-to-end-of-line key
key_eos	ked	kS	KEY_EOS, 0516, by clear-to-end-of-screen key
key_exit	kext	@9	KEY_EXIT, 0551, sent by exit key
key_f0	kf0	k0	KEY_F(0), 0410, sent by function key f0
key_f1	kf1	k1	KEY_F(1), 0411, sent by function key f1

Variable	Cap-name	Termcap Code	Description
key_f2	kf2	k2	KEY_F(2), 0412, sent by function key f2
key_f3	kf3	k3	KEY_F(3), 0413, sent by function key f3
key_f4	kf4	k4	KEY_F(4), 0414, sent by function key f4
key_f5	kf5	k5	KEY_F(5), 0415, sent by function key f5
key_f6	kf6	k6	KEY_F(6), 0416, sent by function key f6
key_f7	kf7	k7	KEY_F(7), 0417, sent by function key f7
key_f8	kf8	k8	KEY_F(8), 0420, sent by function key f8
key_f9	kf9	k9	KEY_F(9), 0421, sent by function key f9
key_f10	kf10	k;	KEY_F(10), 0422, sent by function key f10
key_f11	kf11	F1	KEY_F(11), 0423, sent by function key f11
key_f12	kf12	F2	KEY_F(12), 0424, sent by function key f12
key_f13	kf13	F3	KEY_F(13), 0425, sent by function key f13
key_f14	kf14	F4	KEY_F(14), 0426, sent by function key f14
key_f15	kf15	F5	KEY_F(15), 0427, sent by function key f15
key_f16	kf16	F6	KEY_F(16), 0430, sent by function key f16
key_f17	kf17	F7	KEY_F(17), 0431, sent by function key f17
key_f18	kf18	F8	KEY_F(18), 0432, sent by function key f18
key_f19	kf19	F9	KEY_F(19), 0433, sent by function key f19
key_f20	kf20	FA	KEY_F(20), 0434, sent by function key f20
key_f21	kf21	FB	KEY_F(21), 0435, sent by function key f21
key_f22	kf22	FC	KEY_F(22), 0436, sent by function key f22
key_f23	kf23	FD	KEY_F(23), 0437, sent by function key f23
key_f24	kf24	FE	KEY_F(24), 0440, sent by function key f24
key_f25	kf25	FF	KEY_F(25), 0441, sent by function key f25
key_f26	kf26	FG	KEY_F(26), 0442, sent by function key f26
key_f27	kf27	FH	KEY_F(27), 0443, sent by function key f27
key_f28	kf28	FI	KEY_F(28), 0444, sent by function key f28
key_f29	kf29	FJ	KEY_F(29), 0445, sent by function key f29
key_f30	kf30	FK	KEY_F(30), 0446, sent by function key f30
key_f31	kf31	FL	KEY_F(31), 0447, sent by function key f31
key_f32	kf32	FM	KEY_F(32), 0450, sent by function key f32
key_f33	kf33	FN	KEY_F(13), 0451, sent by function key f33
key_f34	kf34	FO	KEY_F(34), 0452, sent by function key f34
key_f35	kf35	FP	KEY_F(35), 0453, sent by function key f35
key_f36	kf36	FQ	KEY_F(36), 0454, sent by function key f36
key_f37	kf37	FR	KEY_F(37), 0455, sent by function key f37
key_f38	kf38	FS	KEY_F(38), 0456, sent by function key f38
key_f39	kf39	FT	KEY_F(39), 0457, sent by function key f39
key_f40	kf40	FU	KEY_F(40), 0460, sent by function key f40
key_f41	kf41	FV	KEY_F(41), 0461, sent by function key f41
key_f42	kf42	FW	KEY_F(42), 0462, sent by function key f42
key_f43	kf43	FX	KEY_F(43), 0463, sent by function key f43
key_f44	kf44	FY	KEY_F(44), 0464, sent by function key f44
key_f45	kf45	FZ	KEY_F(45), 0465, sent by function key f45

Variable	Cap-name	Termcap Code	Description
key_f46	kf46	Fa	KEY_F(46), 0466, sent by function key f46
key_f47	kf47	Fb	KEY_F(47), 0467, sent by function key f47
key_f48	kf48	Fc	KEY_F(48), 0470, sent by function key f48
key_f49	kf49	Fd	KEY_F(49), 0471, sent by function key f49
key_f50	kf50	Fe	KEY_F(50), 0472, sent by function key f50
key_f51	kf51	Ff	KEY_F(51), 0473, sent by function key f51
key_f52	kf52	Fg	KEY_F(52), 0474, sent by function key f52
key_f53	kf53	Fh	KEY_F(53), 0475, sent by function key f53
key_f54	kf54	Fi	KEY_F(54), 0476, sent by function key f54
key_f55	kf55	Fj	KEY_F(55), 0477, sent by function key f55
key_f56	kf56	Fk	KEY_F(56), 0500, sent by function key f56
key_f57	kf57	Fl	KEY_F(57), 0501, sent by function key f57
key_f58	kf58	Fm	KEY_F(58), 0502, sent by function key f58
key_f59	kf59	Fn	KEY_F(59), 0503, sent by function key f59
key_f60	kf60	Fo	KEY_F(60), 0504, sent by function key f60
key_f61	kf61	Fp	KEY_F(61), 0505, sent by function key f61
key_f62	kf62	Fq	KEY_F(62), 0506, sent by function key f62
key_f63	kf63	Fr	KEY_F(63), 0507, sent by function key f63
key_find	kfnd	@0	KEY_FIND, 0552, sent by find key
key_help	khlp	%1	KEY_HELP, 0553, sent by help key
key_home	khome	kh	KEY_HOME, 0406, sent by home key
key_ic	kich1	kI	KEY_IC, 0513, by ins-char/enter ins-mode key
key_il	kil1	kA	KEY_IL, 0511, sent by insert-line key
key_left	kcub1	kl	KEY_LEFT, 0404, sent by left-arrow key
key_ll	kll	kH	KEY_LL, 0533, sent by home-down key
key_mark	kmrk	%2	KEY_MARK, 0554, sent by mark key
key_message	kmsg	%3	KEY_MESSAGE, 0555, sent by message key
key_move	kmov	%4	KEY_MOVE, 0556, sent by move key
key_next	knxt	%5	KEY_NEXT, 0557, sent by next-object key
key_npage	knp	kN	KEY_NPAGE, 0522, sent by next-page key
key_open	kopn	%6	KEY_OPEN, 0560, sent by open key
key_options	kopt	%7	KEY_OPTIONS, 0561, sent by options key
key_ppage	kpp	kP	KEY_PPAGE, 0523, sent by previous-page key
key_previous	kprv	%8	KEY_PREVIOUS, 0562, by previous-object key
key_print	kprt	%9	KEY_PRINT, 0532, sent by print or copy key
key_redo	krdo	%0	KEY_REDO, 0563, sent by redo key
key_reference	kref	&1	KEY_REFERENCE, 0564, by ref(erence) key
key_refresh	krfr	&2	KEY_REFRESH, 0565, sent by refresh key
key_replace	krpl	&3	KEY_REPLACE, 0566, sent by replace key
key_restart	krst	&4	KEY_RESTART, 0567, sent by restart key
key_resume	kres	&5	KEY_RESUME, 0570, sent by resume key
key_right	kcuf1	kr	KEY_RIGHT, 0405, sent by right-arrow key
key_save	ksav	&6	KEY_SAVE, 0571, sent by save key
key_sbeg	kBEG	&9	KEY_SBEG, 0572, by shifted beginning key

Variable	Cap-name	Termcap Code	Description
key_scancel	kCAN	&0	KEY_SCANCEL, 0573, by shifted cancel key
key_scommand	kCMD	*1	KEY_SCOMMAND,0574, shifted command key
key_scopy	kCPY	*2	KEY_SCOPY, 0575, sent by shifted copy key
key_screate	kCRT	*3	KEY_SCREATE, 0576, by shifted create key
key_sdc	kDC	*4	KEY_SDC, 0577, by shifted delete-char key
key_sdl	kDL	*5	KEY_SDL, 0600, by shifted delete-line key
key_select	kslt	*6	KEY_SELECT, 0601, sent by select key
key_send	kEND	*7	KEY_SEND, 0602, sent by shifted end key
key_seol	kEOL	*8	KEY_SEOL, 0603, sent by shifted clear-line key
key_sexit	kEXT	*9	KEY_SEXIT, 0604, sent by shifted exit key
key_sf	kind	kF	KEY_SF, 0520, sent by scroll-forward/down key
key_sfind	kFND	*0	KEY_SFIND, 0605, sent by shifted find key
key_shelp	kHLP	#1	KEY_SHELP, 0606, sent by shifted help key
key_shome	kHOM	#2	KEY_SHOME, 0607, sent by shifted home key
key_sic	kIC	#3	KEY_SIC, 0610, sent by shifted input key
key_sleft	kLFT	#4	KEY_SLEFT, 0611, by shifted left-arrow key
key_smessage	kMSG	%a	KEY_SMESSAGE, 0612, by shifted message key
key_smove	kMOV	%b	KEY_SMOVE, 0613, sent by shifted move key
key_snext	kNXT	%c	KEY_SNEXT, 0614, sent by shifted next key
key_soptions	kOPT	%d	KEY_SOPTIONS, 0615, by shifted options key
key_sprevious	kPRV	%e	KEY_SPREVIOUS, 0616, by shifted prev key
key_sprint	kPRT	%f	KEY_SPRINT, 0617, by shifted print key
key_sr	kri	kR	KEY_SR, 0521, sent by scroll-backward/up key
key_sredo	kRDO	%g	KEY_SREDO, 0620, sent by shifted redo key
key_sreplace	kRPL	%h	KEY_SREPLACE, 0621, by shifted replace key
key_sright	kRIT	%i	KEY_SRIGHT, 0622, by shifted right-arrow key
key_srsume	kRES	%j	KEY_SRSUME, 0623, by shifted resume key
key_ssave	kSAV	!1	KEY_SSAVE, 0624, sent by shifted save key
key_ssuspend	kSPD	!2	KEY_SSUSPEND, 0625, by shifted suspend key
key_stab	khts	kT	KEY_STAB, 0524, sent by set-tab key
key_sundo	kUND	!3	KEY_SUNDO, 0626, sent by shifted undo key
key_suspend	kspd	&7	KEY_SUSPEND, 0627, sent by suspend key
key_undo	kund	&8	KEY_UNDO, 0630, sent by undo key
key_up	kcuu1	ku	KEY_UP, 0403, sent by up-arrow key
keypad_local	rmkx	ke	Out of "keypad-transmit" mode
keypad_xmit	smkx	ks	Put terminal in "keypad-transmit" mode
lab_f0	lf0	l0	Labels on function key f0 if not f0
lab_f1	lf1	l1	Labels on function key f1 if not f1
lab_f2	lf2	l2	Labels on function key f2 if not f2
lab_f3	lf3	l3	Labels on function key f3 if not f3
lab_f4	lf4	l4	Labels on function key f4 if not f4
lab_f5	lf5	l5	Labels on function key f5 if not f5
lab_f6	lf6	l6	Labels on function key f6 if not f6
lab_f7	lf7	l7	Labels on function key f7 if not f7

Variable	Cap-name	Termcap Code	Description
lab_f8	lf8	l8	Labels on function key f8 if not f8
lab_f9	lf9	l9	Labels on function key f9 if not f9
lab_f10	lf10	la	Labels on function key f10 if not f10
label_off	rmln	LF	Turn off soft labels
label_on	smln	LO	Turn on soft labels
meta_off	rmm	mo	Turn off "meta mode"
meta_on	smm	mm	Turn on "meta mode" (8th bit)
micro_column_address	mhpa	ZY	Like **column_address** for micro adjustment
micro_down	mcud1	ZZ	Like **cursor_down** for micro adjustment
micro_left	mcub1	Za	Like **cursor_left** for micro adjustment
micro_right	mcuf1	Zb	Like **cursor_right** for micro adjustment
micro_row_address	mvpa	Zc	Like **row_address** for micro adjustment
micro_up	mcuu1	Zd	Like **cursor_up** for micro adjustment
newline	nel	nw	Newline (behaves like cr followed by lf)
order_of_pins	porder	Ze	Matches software bits to print-head pins
orig_colors	oc	oc	Set all color(-pair)s to the original ones
orig_pair	op	op	Set default color-pair to the original one
pad_char	pad	pc	Pad character (rather than null)
parm_dch	dch	DC	Delete #1 chars
parm_delete_line	dl	DL	Delete #1 lines
parm_down_cursor	cud	DO	Move down #1 lines.
parm_down_micro	mcud	Zf	Like **parm_down_cursor** for micro adjust.
parm_ich	ich	IC	Insert #1 blank chars
parm_index	indn	SF	Scroll forward #1 lines.
parm_insert_line	il	AL	Add #1 new blank lines
parm_left_cursor	cub	LE	Move cursor left #1 spaces
parm_left_micro	mcub	Zg	Like **parm_left_cursor** for micro adjust.
parm_right_cursor	cuf	RI	Move right #1 spaces.
parm_right_micro	mcuf	Zh	Like **parm_right_cursor** for micro adjust.
parm_rindex	rin	SR	Scroll backward #1 lines.
parm_up_cursor	cuu	UP	Move cursor up #1 lines.
parm_up_micro	mcuu	Zi	Like **parm_up_cursor** for micro adjust.
pkey_key	pfkey	pk	Prog funct key #1 to type string #2
pkey_local	pfloc	pl	Prog funct key #1 to execute string #2
pkey_xmit	pfx	px	Prog funct key #1 to xmit string #2
plab_norm	pln	pn	Prog label #1 to show string #2
print_screen	mc0	ps	Print contents of the screen
prtr_non	mc5p	pO	Turn on the printer for #1 bytes
prtr_off	mc4	pf	Turn off the printer
prtr_on	mc5	po	Turn on the printer
repeat_char	rep	rp	Repeat char #1 #2 times
req_for_input	rfi	RF	Send next input char (for ptys)
reset_1string	rs1	r1	Reset terminal completely to sane modes
reset_2string	rs2	r2	Reset terminal completely to sane modes

Variable	Cap-name	Termcap Code	Description
reset_3string	rs3	r3	Reset terminal completely to sane modes
reset_file	rf	rf	Name of file containing reset string
restore_cursor	rc	rc	Restore cursor to position of last sc
row_address	vpa	cv	Vertical position absolute
save_cursor	sc	sc	Save cursor position
scroll_forward	ind	sf	Scroll text up
scroll_reverse	ri	sr	Scroll text down
select_char_set	scs	Zj	Select character set
set_attributes	sgr	sa	Define the video attributes #1-#9
set_background	setb	Sb	Set current background color
set_bottom_margin	smgb	Zk	Set bottom margin at current line
set_bottom_margin_parm	smgbp	Zl	Set bottom margin #2 lines from bottom
set_color_pair	scp	sp	Set current color-pair
set_foreground	setf	Sf	Set current foreground color1
set_left_margin	smgl	ML	Set left margin at current line
set_left_margin_parm	smglp	Zm	Set left (right) margin at column #1 (#2)
set_right_margin	smgr	MR	Set right margin at current column
set_right_margin_parm	smgrp	Zn	Set right margin at column #1
set_tab	hts	st	Set a tab in all rows, current column
set_top_margin	smgt	Zo	Set top margin at current line
set_top_margin_parm	smgtp	Zp	Set top (bottom) margin at line #1 (#2)
set_window	wind	wi	Current window is lines #1-#2 cols #3-#4
start_bit_image	sbim	Zq	Start printing bit image graphics
start_char_set_def	scsd	Zr	Start definition of a character set
stop_bit_image	rbim	Zs	End printing bit image graphics
stop_char_set_def	rcsd	Zt	End definition of a character set
subscript_characters	subcs	Zu	List of "subscript-able" characters
superscript_characters	supcs	Zv	List of "superscript-able" characters
tab	ht	ta	Tab to next 8-space hardware tab stop
these_cause_cr	docr	Zw	Printing any of these chars causes cr
to_status_line	tsl	ts	Go to status line, col #1
underline_char	uc	uc	Underscore one char and move past it
up_half_line	hu	hu	Half-line up (reverse 1/2 linefeed)
xoff_character	xoffc	XF	X-off character
xon_character	xonc	XN	X-on character
zero_motion	zerom	Zx	No motion for the subsequent character

Booleans

Cap-name	Variable	Termcap Code	Description
am	auto_right_margin	am	Terminal has automatic margins
bw	auto_left_margin	bw	**cub1** wraps from column 0 to last column
ccc	can_change	cc	Terminal can re-define existing color
chts	hard_cursor	HC	Cursor is hard to see

Cap-name	Variable	Termcap Code	Description
cpix	cpi_changes_res	YF	Changing character pitch changes resolution
crxm	cr_cancels_micro_modem	YB	Using **cr** turns off micro mode
da	memory_above	da	Display may be retained above the screen
daisy	has_print_wheel	YC	Printer needs operator to change character set
db	memory_below	db	Display may be retained below the screen
eo	erase_overstrike	eo	Can erase overstrikes with a blank
eslok	status_line_esc_ok	es	Escape can be used on the status line
gn	generic_type	gn	Generic line type (e.g., dialup, switch)
hc	hard_copy	hc	Hardcopy terminal
hls	hue_lightness_saturation	hl	Terminal uses only HLS color notation (Tektronix)
hs	has_status_line	hs	Has extra "status line"
hz	tilde_glitch	hz	Hazeltine; can't print tilde (~)
in	insert_null_glitch	in	Insert mode distinguishes nulls
km	has_meta_key	km	Has a meta key (shift, sets parity bit)
lpix	lpi_changes_res	YG	Changing line pitch changes resolution
mc5i	prtr_silent	5i	Printer won't echo on screen
mir	move_insert_mode	mi	Safe to move while in insert mode
msgr	move_standout_mode	ms	Safe to move in standout modes
npc	no_pad_char	NP	Pad character doesn't exist
nrrmc	non_rev_rmcup	NR	**smcup** does not reverse **rmcup**
nxon	needs_xon_xoff	nx	Padding won't work, xon/xoff required
os	over_strike	os	Terminal overstrikes on hard-copy terminal
sam	semi_auto_right_margin	YE	Printing in last column causes **cr**
ul	transparent_underline	ul	Underline character overstrikes
xenl	eat_newline_glitch	xn	Newline ignored after 80 columns (*Concept*)
xhp	ceol_standout_glitch	xs	Standout not erased by overwriting (hp)
xhpa	col_addr_glitch	YA	Only positive motion for **hpa/mhpa** caps
xon	xon_xoff	xo	Terminal uses xon/xoff handshaking
xsb	no_esc_ctlc	xb	Beehive (f1=escape, f2=ctrl C)
xt	dest_tabs_magic_smso	xt	Destructive tabs, magic **smso** char (t1061)
xvpa	row_addr_glitch	YD	Only positive motion for **vpa/mvpa** caps

Numbers

Cap-name	Variable	Termcap Code	Description
bufsz	buffer_capacity	Ya	Number of bytes buffered before printing
colors	max_colors	Co	Maximum number of colors on the screen
cols	columns	co	Number of columns in a line
cps	print_rate	Ym	Average print rate in characters per second
it	init_tabs	it	Tabs initially every # spaces
lh	label_height	lh	Number of rows in each label
lines	lines	li	Number of lines on a screen or a page
lm	lines_of_memory	lm	Lines of memory if > **lines**; 0 means varies
lw	label_width	lw	Number of columns in each label

Cap-name	Variable	Termcap Code	Description
maddr	max_micro_address	Yd	Maximum value in **micro_..._address**
mcs	micro_col_size	Yf	Character step size when in micro mode
mjump	max_micro_jump	Ye	Maximum value in **parm_..._micro**
mls	micro_line_size	Yg	Line step size when in micro mode
ncv	no_color_video	NC	Video attributes that can't be used with colors
nlab	num_labels	Nl	Number of labels on screen (start at 1)
npins	number_of_pins	Yh	Number of pins in print-head
orc	output_res_char	Yi	Horizontal resolution in units per character
orhi	output_res_horz_inch	Yk	Horizontal resolution in units per inch
orl	output_res_line	Yj	Vertical resolution in units per line
orvi	output_res_vert_inch	Yl	Vertical resolution in units per inch
pairs	max_pairs	pa	Maximum number of color-pairs on the screen
pb	padding_baud_rate	pb	Lowest baud rate where padding needed
spinh	dot_horz_spacing	Yc	Spacing of dots horizontally in dots per inch
spinv	dot_vert_spacing	Yb	Spacing of pins vertically in pins per inch
vt	virtual_terminal	vt	Virtual terminal number (DYNIX/ptx)
widcs	wide_char_size	Yn	Character step size when in double wide mode
wsl	width_status_line	ws	Number of columns in status line
xmc	magic_cookie_glitch	sg	Number of blank characters left by **smso** or **rmso**

Strings

Cap-name	Variable	Termcap Code	Description
acsc	acs_chars	ac	Graphic charset pairs aAbBcC - def=vt100
bel	bell	bl	Audible signal (bell)
blink	enter_blink_mode	mb	Turn on blinking
bold	enter_bold_mode	md	Turn on bold (extra bright) mode
cbt	back_tab	bt	Back tab
chr	change_res_horz	ZC	Change horizontal resolution
civis	cursor_invisible	vi	Make cursor invisible
clear	clear_screen	cl	Clear screen and home cursor
cmdch	command_character	CC	Terminal settable cmd character in prototype
cnorm	cursor_normal	ve	Make cursor appear normal (undo **vs**/**vi**)
cpi	change_char_pitch	ZA	Change number of characters per inch
cr	carriage_return	cr	Carriage return
csnm	char_set_names	Zy	List of character set names
csr	change_scroll_region	cs	Change to lines #1 through #2 (vt100)
cub	parm_left_cursor	LE	Move cursor left #1 spaces
cub1	cursor_left	le	Move left one space.
cud	parm_down_cursor	DO	Move down #1 lines.
cuf	parm_right_cursor	RI	Move right #1 spaces.
cuf1	cursor_right	nd	Non-destructive space (cursor or carriage right)
cup	cursor_address	cm	Move to row #1 col #2
cuu	parm_up_cursor	UP	Move cursor up #1 lines.

Cap-name	Variable	Termcap Code	Description
cvr	change_res_vert	ZD	Change vertical resolution
cvvis	cursor_visible	vs	Make cursor very visible
dch	parm_dch	DC	Delete #1 chars
dch1	delete_character	dc	Delete character
defc	define_char	ZE	Define a character in a character set
dim	enter_dim_mode	mh	Turn on half-bright mode
dl	delete_line	dl1	Delete line
dl	parm_delete_line	DL	Delete #1 lines
do	cursor_down	do	Down one line
docr	these_cause_cr	Zw	Printing any of these chars causes cr
dsl	dis_status_line	ds	Disable status line
ech	erase_chars	ec	Erase #1 characters
ed	clr_eos	cd	Clear to end of display
el	clr_eol	ce	Clear to end of line
el1	clr_bol	cb	Clear to beginning of line, inclusive
enacs	ena_acs	eA	Enable alternate character set
ff	form_feed	ff	Hardcopy terminal page eject
flash	flash_screen	vb	Visible bell (may not move cursor)
fsl	from_status_line	fs	Return from status line
hd	down_half_line	hd	Half-line down (forward 1/2 linefeed)
home	cursor_home	ho	Home cursor (if no **cup**)
hpa	column_address	ch	Horizontal position absolute
ht	tab	ta	Tab to next 8-space hardware tab stop
hts	set_tab	st	Set a tab in all rows, current column
hu	up_half_line	hu	Half-line up (reverse 1/2 linefeed)
ich	parm_ich	IC	Insert #1 blank chars
ich1	insert_character	ic	Insert character
if	init_file	if	Name of initialization file
il	parm_insert_line	AL	Add #1 new blank lines
il1	insert_line	al	Add new blank line
ind	scroll_forward	sf	Scroll text up
indn	parm_index	SF	Scroll forward #1 lines.
initc	initialize_color	Ic	Initialize the definition of color
initp	initialize_pair	Ip	Initialize color-pair
invis	enter_secure_mode	mk	Turn on blank mode (characters invisible)
ip	insert_padding	ip	Insert pad after character inserted
iprog	init_prog	iP	Path name of program for initialization
is1	init_1string	i1	Terminal or printer initialization string
is2	init_2string	is	Terminal or printer initialization string
is3	init_3string	i3	Terminal or printer initialization string
kBEG	key_sbeg	&9	KEY_SBEG, 0572, sent by shifted beginning key
kCAN	key_scancel	&0	KEY_SCANCEL, 0573, sent by shifted cancel key
kCMD	key_scommand	*1	KEY_SCOMMAND, 0574, sent by shifted command key
kCPY	key_scopy	*2	KEY_SCOPY, 0575, sent by shifted copy key

Cap-name	Variable	Termcap Code	Description
kCRT	key_screate	*3	KEY_SCREATE, 0576, sent by shifted create key
kDC	key_sdc	*4	KEY_SDC, 0577, sent by shifted delete-char key
kDL	key_sdl	*5	KEY_SDL, 0600, sent by shifted delete-line key
kEND	key_send	*7	KEY_SEND, 0602, sent by shifted end key
kEOL	key_seol	*8	KEY_SEOL, 0603, sent by shifted clear-line key
kEXT	key_sexit	*9	KEY_SEXIT, 0604, sent by shifted exit key
kFND	key_sfind	*0	KEY_SFIND, 0605, sent by shifted find key
kHLP	key_shelp	#1	KEY_SHELP, 0606, sent by shifted help key
kHOM	key_shome	#2	KEY_SHOME, 0607, sent by shifted home key
kIC	key_sic	#3	KEY_SIC, 0610, sent by shifted input key
kLFT	key_sleft	#4	KEY_SLEFT, 0611, sent by shifted left-arrow key
kMOV	key_smove	%b	KEY_SMOVE, 0613, sent by shifted move key
kMSG	key_smessage	%a	KEY_SMESSAGE, 0612, sent by shifted message key
kNXT	key_snext	%c	KEY_SNEXT, 0614, sent by shifted next key
kOPT	key_soptions	%d	KEY_SOPTIONS, 0615, sent by shifted options key
kPRT	key_sprint	%f	KEY_SPRINT, 0617, sent by shifted print key
kPRV	key_sprevious	%e	KEY_SPREVIOUS, 0616, sent by shifted prev key
kRDO	key_sredo	%g	KEY_SREDO, 0620, sent by shifted redo key
kRES	key_srsume	%j	KEY_SRSUME, 0623, sent by shifted resume key
kRIT	key_sright	%i	KEY_SRIGHT, 0622, sent by shifted right-arrow key
kRPL	key_sreplace	%h	KEY_SREPLACE, 0621, sent by shifted replace key
kSAV	key_ssave	!1	KEY_SSAVE, 0624, sent by shifted save key
kSPD	key_ssuspend	!2	KEY_SSUSPEND, 0625, sent by shifted suspend key
kUND	key_sundo	!3	KEY_SUNDO, 0626, sent by shifted undo key
ka1	key_a1	K1	KEY_A1, 0534, upper left of keypad
ka3	key_a3	K3	KEY_A3, 0535, upper right of keypad
kb2	key_b2	K2	KEY_B2, 0536, center of keypad
kbeg	key_beg	@1	KEY_BEG, 0542, sent by beg(inning) key
kbs	key_backspace	kb	KEY_BACKSPACE, 0407, sent by backspace key
kc1	key_c1	K4	KEY_C1, 0537, lower left of keypad
kc3	key_c3	K5	KEY_C3, 0540, lower right of keypad
kcan	key_cancel	@2	KEY_CANCEL, 0543, sent by cancel key
kcbt	key_btab	kB	KEY_BTAB, 0541, sent by back-tab key
kclo	key_close	@3	KEY_CLOSE, 0544, sent by close key
kclr	key_clear	kC	KEY_CLEAR, 0515, sent by clear-screen or erase key
kcmd	key_command	@4	KEY_COMMAND, 0545, sent by cmd (command) key
kcpy	key_copy	@5	KEY_COPY, 0546, sent by copy key
kcrt	key_create	@6	KEY_CREATE, 0547, sent by create key
kctab	key_ctab	kt	KEY_CTAB, 0525, sent by clear-tab key
kcub1	key_left	kl	KEY_LEFT, 0404, sent by terminal left-arrow key
kcud1	key_down	kd	KEY_DOWN, 0402, sent by terminal down-arrow key
kcuf1	key_right	kr	KEY_RIGHT, 0405, sent by terminal right-arrow key
kcuu1	key_up	ku	KEY_UP, 0403, sent by terminal up-arrow key
kdch1	key_dc	kD	KEY_DC, 0512, sent by delete-character key

Cap-name	Variable	Termcap Code	Description
kdl1	key_dl	kL	KEY_DL, 0510, sent by delete-line key
ked	key_eos	ked	KEY_EOS, 0516, sent by clear-to-end-of-screen key
kel	key_eol	kE	KEY_EOL, 0517, sent by clear-to-end-of-line key
kend	key_end	@7	KEY_END, 0550, sent by end key
kent	key_enter	@8	KEY_ENTER, 0527, sent by enter/send key
kext	key_exit	@9	KEY_EXIT, 0551, sent by exit key
kf0	key_f0	k0	KEY_F(0), 0410, sent by function key f0
kf1	key_f1	k1	KEY_F(1), 0411, sent by function key f1
kf10	key_f10	k;	KEY_F(10), 0422, sent by function key f10
kf11	key_f11	F1	KEY_F(11), 0423, sent by function key f11
kf12	key_f12	F2	KEY_F(12), 0424, sent by function key f12
kf13	key_f13	F3	KEY_F(13), 0425, sent by function key f13
kf14	key_f14	F4	KEY_F(14), 0426, sent by function key f14
kf15	key_f15	F5	KEY_F(15), 0427, sent by function key f15
kf16	key_f16	F6	KEY_F(16), 0430, sent by function key f16
kf17	key_f17	F7	KEY_F(17), 0431, sent by function key f17
kf18	key_f18	F8	KEY_F(18), 0432, sent by function key f18
kf19	key_f19	F9	KEY_F(19), 0433, sent by function key f19
kf2	key_f2	k2	KEY_F(2), 0412, sent by function key f2
kf20	key_f20	FA	KEY_F(20), 0434, sent by function key f20
kf21	key_f21	FB	KEY_F(21), 0435, sent by function key f21
kf22	key_f22	FC	KEY_F(22), 0436, sent by function key f22
kf23	key_f23	FD	KEY_F(23), 0437, sent by function key f23
kf24	key_f24	FE	KEY_F(24), 0440, sent by function key f24
kf25	key_f25	FF	KEY_F(25), 0441, sent by function key f25
kf26	key_f26	FG	KEY_F(26), 0442, sent by function key f26
kf27	key_f27	FH	KEY_F(27), 0443, sent by function key f27
kf28	key_f28	FI	KEY_F(28), 0444, sent by function key f28
kf29	key_f29	FJ	KEY_F(29), 0445, sent by function key f29
kf3	key_f3	k3	KEY_F(3), 0413, sent by function key f3
kf30	key_f30	FK	KEY_F(30), 0446, sent by function key f30
kf31	key_f31	FL	KEY_F(31), 0447, sent by function key f31
kf32	key_f32	FM	KEY_F(32), 0450, sent by function key f32
kf33	key_f33	FN	KEY_F(13), 0451, sent by function key f33
kf34	key_f34	FO	KEY_F(34), 0452, sent by function key f34
kf35	key_f35	FP	KEY_F(35), 0453, sent by function key f35
kf36	key_f36	FQ	KEY_F(36), 0454, sent by function key f36
kf37	key_f37	FR	KEY_F(37), 0455, sent by function key f37
kf38	key_f38	FS	KEY_F(38), 0456, sent by function key f38
kf39	key_f39	FT	KEY_F(39), 0457, sent by function key f39
kf4	key_f4	k4	KEY_F(4), 0414, sent by function key f4
kf40	key_f40	FU	KEY_F(40), 0460, sent by function key f40
kf41	key_f41	FV	KEY_F(41), 0461, sent by function key f41
kf42	key_f42	FW	KEY_F(42), 0462, sent by function key f42

Cap-name	Variable	Termcap Code	Description
kf43	key_f43	FX	KEY_F(43), 0463, sent by function key f43
kf44	key_f44	FY	KEY_F(44), 0464, sent by function key f44
kf45	key_f45	FZ	KEY_F(45), 0465, sent by function key f45
kf46	key_f46	Fa	KEY_F(46), 0466, sent by function key f46
kf47	key_f47	Fb	KEY_F(47), 0467, sent by function key f47
kf48	key_f48	Fc	KEY_F(48), 0470, sent by function key f48
kf49	key_f49	Fd	KEY_F(49), 0471, sent by function key f49
kf5	key_f5	k5	KEY_F(5), 0415, sent by function key f5
kf50	key_f50	Fe	KEY_F(50), 0472, sent by function key f50
kf51	key_f51	Ff	KEY_F(51), 0473, sent by function key f51
kf52	key_f52	Fg	KEY_F(52), 0474, sent by function key f52
kf53	key_f53	Fh	KEY_F(53), 0475, sent by function key f53
kf54	key_f54	Fi	KEY_F(54), 0476, sent by function key f54
kf55	key_f55	Fj	KEY_F(55), 0477, sent by function key f55
kf56	key_f56	Fk	KEY_F(56), 0500, sent by function key f56
kf57	key_f57	Fl	KEY_F(57), 0501, sent by function key f57
kf58	key_f58	Fm	KEY_F(58), 0502, sent by function key f58
kf59	key_f59	Fn	KEY_F(59), 0503, sent by function key f59
kf6	key_f6	k6	KEY_F(6), 0416, sent by function key f6
kf60	key_f60	Fo	KEY_F(60), 0504, sent by function key f60
kf61	key_f61	Fp	KEY_F(61), 0505, sent by function key f61
kf62	key_f62	Fq	KEY_F(62), 0506, sent by function key f62
kf63	key_f63	Fr	KEY_F(63), 0507, sent by function key f63
kf7	key_f7	k7	KEY_F(7), 0417, sent by function key f7
kf8	key_f8	k8	KEY_F(8), 0420, sent by function key f8
kf9	key_f9	k9	KEY_F(9), 0421, sent by function key f9
kfnd	key_find	@0	KEY_FIND, 0552, sent by find key
khlp	key_help	%1	KEY_HELP, 0553, sent by help key
khome	key_home	kh	KEY_HOME, 0406, sent by home key
khts	key_stab	kT	KEY_STAB, 0524, sent by set-tab key
kich1	key_ic	kI	KEY_IC, 0513, sent by ins-char/enter ins-mode key
kil1	key_il	kA	KEY_IL, 0511, sent by insert-line key
kind	key_sf	kF	KEY_SF, 0520, sent by scroll-forward/down key
kll	key_ll	kH	KEY_LL, 0533, sent by home-down key
kmov	key_move	%4	KEY_MOVE, 0556, sent by move key
kmrk	key_mark	%2	KEY_MARK, 0554, sent by mark key
kmsg	key_message	%3	KEY_MESSAGE, 0555, sent by message key
knp	key_npage	kN	KEY_NPAGE, 0522, sent by next-page key
knxt	key_next	%5	KEY_NEXT, 0557, sent by next-object key
kopn	key_open	%6	KEY_OPEN, 0560, sent by open key
kopt	key_options	%7	KEY_OPTIONS, 0561, sent by options key
kpp	key_ppage	kP	KEY_PPAGE, 0523, sent by previous-page key
kprt	key_print	%9	KEY_PRINT, 0532, sent by print or copy key
kprv	key_previous	%8	KEY_PREVIOUS, 0562, sent by previous-object key

Cap-name	Variable	Termcap Code	Description
krdo	key_redo	%0	KEY_REDO, 0563, sent by redo key
kref	key_reference	&1	KEY_REFERENCE, 0564, sent by ref(erence) key
kres	key_resume	&5	KEY_RESUME, 0570, sent by resume key
krfr	key_refresh	&2	KEY_REFRESH, 0565, sent by refresh key
kri	key_sr	kR	KEY_SR, 0521, sent by scroll-backward/up key
krmir	key_eic	kM	KEY_EIC, 0514, sent by **rmir** or **smir** in insert mode
krpl	key_replace	&3	KEY_REPLACE, 0566, sent by replace key
krst	key_restart	&4	KEY_RESTART, 0567, sent by restart key
ksav	key_save	&6	KEY_SAVE, 0571, sent by save key
kslt	key_select	*6	KEY_SELECT, 0601, sent by select key
kspd	key_suspend	&7	KEY_SUSPEND, 0627, sent by suspend key
ktbc	key_catab	ka	KEY_CATAB, 0526, sent by clear-all-tabs key
kund	key_undo	&8	KEY_UNDO, 0630, sent by undo key
lf0	lab_f0	l0	Labels on function key f0 if not f0
lf1	lab_f1	l1	Labels on function key f1 if not f1
lf10	lab_f10	la	Labels on function key f10 if not f10
lf2	lab_f2	l2	Labels on function key f2 if not f2
lf3	lab_f3	l3	Labels on function key f3 if not f3
lf4	lab_f4	l4	Labels on function key f4 if not f4
lf5	lab_f5	l5	Labels on function key f5 if not f5
lf6	lab_f6	l6	Labels on function key f6 if not f6
lf7	lab_f7	l7	Labels on function key f7 if not f7
lf8	lab_f8	l8	Labels on function key f8 if not f8
lf9	lab_f9	l9	Labels on function key f9 if not f9
ll	cursor_to_ll	ll	Last line, first column (if no **cup**)
lpi	change_line_pitch	ZB	Change number of lines per inch
mc0	print_screen	ps	Print contents of the screen
mc4	prtr_off	pf	Turn off the printer
mc5	prtr_on	po	Turn on the printer
mc5p	prtr_non	pO	Turn on the printer for #1 bytes
mcub	parm_left_micro	Zg	Like **parm_left_cursor** for micro adjust.
mcub1	micro_left	Za	Like **cursor_left** for micro adjustment
mcud	parm_down_micro	Zf	Like **parm_down_cursor** for micro adjust.
mcud1	micro_down	ZZ	Like **cursor_down** for micro adjustment
mcuf	parm_right_micro	Zh	Like **parm_right_cursor** for micro adjust.
mcuf1	micro_right	Zb	Like **cursor_right** for micro adjustment
mcuu	parm_up_micro	Zi	Like **parm_up_cursor** for micro adjust.
mcuu1	micro_up	Zd	Like **cursor_up** for micro adjustment)
mgc	clear_margins	MC	Clear all margins (top, bottom, and sides)
mhpa	micro_column_address	ZY	Like **column_address** for micro adjustment
mrcup	cursor_mem_address	CM	Memory relative cursor addressing
mvpa	micro_row_address	Zc	Like **row_address** for micro adjustment
nel	newline	nw	Newline (behaves like **cr** followed by **lf**)
oc	orig_colors	oc	Set all color(-pair)s to the original ones

Cap-name	Variable	Termcap Code	Description
op	orig_pair	op	Set default color-pair to the original one
pad	pad_char	pc	Pad character (rather than null)
pfkey	pkey_key	pk	Prog funct key #1 to type string #2
pfloc	pkey_local	pl	Prog funct key #1 to execute string #2
pfx	pkey_xmit	px	Prog funct key #1 to xmit string #2
pln	plab_norm	pn	Prog label #1 to show string #2
porder	order_of_pins	Ze	Matches software bits to print-head pins
prot	enter_protected_mode	mp	Turn on protected mode
rbim	stop_bit_image	Zs	End printing bit image graphics
rc	restore_cursor	rc	Restore cursor to position of last sc
rcsd	stop_char_set_def	Zt	End definition of a character set
rep	repeat_char	rp	Repeat char #1 #2 times
rev	enter_reverse_mode	mr	Turn on reverse video mode
rf	reset_file	rf	Name of file containing reset string
rfi	req_for_input	RF	Send next input char (for ptys)
ri	scroll_reverse	sr	Scroll text down
rin	parm_rindex	SR	Scroll backward #1 lines.
ritm	exit_italics_mode	ZR	Disable italics
rlm	exit_leftward_mode	ZS	Enable rightward (normal) carriage motion
rmacs	exit_alt_charset_mode	ae	End alternate character set
rmam	exit_am_mode	RA	Turn off automatic margins
rmcup	exit_ca_mode	te	String to end programs that use **cup**
rmdc	exit_delete_mode	ed	End delete mode
rmicm	exit_micro_mode	ZT	Disable micro motion capabilities
rmir	exit_insert_mode	ei	End insert mode
rmkx	keypad_local	ke	Out of "keypad-transmit" modey
rmln	label_off	LF	Turn off soft labels
rmm	meta_off	mo	Turn off "meta mode"
rmp	char_padding	rP	Like **ip** but when in replace mode
rmso	exit_standout_mode	se	End standout mode
rmul	exit_underline_mode	ue	End underscore mode
rmxon	exit_xon_mode	RX	Turn off xon/xoff handshaking
rs1	reset_1string	r1	Reset terminal completely to sane modes
rs2	reset_2string	r2	Reset terminal completely to sane modes
rs3	reset_3string	r3	Reset terminal completely to sane modes
rshm	exit_shadow_mode	ZU	Disable shadow printing
rsubm	exit_subscript_mode	ZV	Disable subscript printing
rsupm	exit_superscript_mode	ZW	Disable superscript printing
rum	exit_upward_mode	ZX	Enable downward (normal) carriage motion
rwidm	exit_doublewide_mode	ZQ	Disable double wide printing
sbim	start_bit_image	Zq	Start printing bit image graphics
sc	save_cursor	sc	Save cursor position
scp	set_color_pair	sp	Set current color-pair
scs	select_char_set	Zj	Select character set

Cap-name	Variable	Termcap Code	Description
scsc	start_char_set_def	Zr	Start definition of a character set
sdrfq	enter_draft_quality	ZG	Set draft quality print
setb	set_background	Sb	Set current background color
setf	set_foreground	Sf	Set current foreground color
sgr	set_attributes	sa	Define the video attributes #1-#9
sgr0	exit_attribute_mode	me	Turn off all attributes
sitm	enter_italics_mode	ZH	Enable italics
slm	enter_leftward_mode	ZI	Enable leftward carriage motion
smacs	enter_alt_charset_mode	as	Start alternate character set
smam	enter_am_mode	SA	Turn on automatic margins
smcup	enter_ca_mode	ti	String to begin programs that use **cup**
smdc	enter_delete_mode	dm	Delete mode (enter)
smgb	set_bottom_margin	Zk	Set bottom margin at current line
smgbp	set_bottom_margin_parm	Zl	Set bottom margin at line #1 or #2 lines from bottom
smgl	set_left_margin	ML	Set left margin at current line
smglp	set_left_margin_parm	Zm	Set left (right) margin at column #1 (#2)
smgr	set_right_margin	MR	Set right margin at current column
smgrp	set_right_margin_parm	Zn	Set right margin at column #1
smgt	set_top_margin	Zo	Set top margin at current line
smgtp	set_top_margin_parm	Zp	Set top (bottom) margin at line #1 (#2)
smicm	enter_micro_mode	ZJ	Enable micro motion capabilities
smir	enter_insert_mode	im	Insert mode (enter)
smkx	keypad_xmit	ks	Put terminal in "keypad-transmit" mode
smln	label_on	LO	Turn on soft labels
smm	meta_on	mm	Turn on "meta mode" (8th bit)
smso	enter_standout_mode	so	Begin standout mode
smul	enter_underline_mode	us	Start underscore mode
smxon	enter_xon_mode	SX	Turn on xon/xoff handshaking
snlq	enter_near_letter_quality	ZK	Set near-letter quality print
snrmq	enter_normal_quality	ZL	Set normal quality print
sshm	enter_shadow_mode	ZM	Enable shadow printing
ssubm	enter_subscript_mode	ZN	Enable subscript printing
ssupm	enter_superscript_mode	ZO	Enable superscript printing
subcs	subscript_characters	Zu	List of "subscript-able" characters
sum	enter_upward_mode	ZP	Enable upward carriage motion
supcs	superscript_characters	Zv	List of "superscript-able" characters
swidm	enter_doublewide_mode	ZF	Enable double wide printing
tbc	clear_all_tabs	ct	Clear all tab stops
tsl	to_status_line	ts	Go to status line, col #1
uc	underline_char	uc	Underscore one char and move past it
up	cursor_up	cuu1	Upline (cursor up)
vpa	row_address	cv	Vertical position absolute
wind	set_window	wi	Current window is lines #1-#2 cols #3-#4
xoffc	xoff_character	XF	X-off character

Cap-name	Variable	Termcap Code	Description
xonc	xon_character	XN	X-on character
zerom	zero_motion	Zx	No motion for the subsequent character

EXAMPLE

The following entry, which describes the AT&T 610 terminal, is one of the most complex entries in the *terminfo* file.

```
610 | 610bct | ATT610 | att610 | AT&T 610; 80 column; 98key keyboard
    am, eslok, hs, mir, msgr, xenl, xon,
    cols#80, it#8, lh#2, lines#24, lw#8, nlab#8, wsl#80,
    acsc=``aaffggjjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{ | | }}~~,
    bel=^G, blink=\E[5m, bold=\E[1m, cbt=\E[Z,
    civis=\E[?25l, clear=\E[H\E[J, cnorm=\E[?25h\E[?12l,
    cr=\r, csr=\E[%i%p1%d;%p2%dr, cub=\E[%p1%dD, cub1=\b,
    cud=\E[%p1%dB, cud1=\E[B, cuf=\E[%p1%dC, cuf1=\E[C,
    cup=\E[%i%p1%d;%p2%dH, cuu=\E[%p1%dA, cuu1=\E[A,
    cvvis=\E[?12;25h, dch=\E[%p1%dP, dch1=\E[P, dim=\E[2m,
    dl=\E[%p1%dM, dl1=\E[M, ed=\E[J, el=\E[K, el1=\E[1K,
    flash=\E[?5h$<200>\E[?5l, fsl=\E8, home=\E[H, ht=\t,
    ich=\E[%p1%d@, il=\E[%p1%dL, il1=\E[L, ind=\ED,
    invis=\E[8m,
    is1=\E[8;0 | \E[?3;4;5;13;15l\E[13;20l\E[?7h\E[12h\E(B\E)0,
    is2=\E[0m^O, is3=\E(B\E)0, kLFT=\E[\s@, kRIT=\E[\sA,
    kbs=\b, kcbt=\E[Z, kclr=\E[2J, kcub1=\E[D, kcud1=\E[B,
    kcuf1=\E[C, kcuu1=\E[A, kf1=\EOc, kf10=\ENp,
    kf11=\ENq, kf12=\ENr, kf13=\ENs, kf14=\ENt, kf2=\EOd,
    kf3=\EOe, kf4=\EOf, kf5=\EOg, kf6=\EOh, kf7=\EOi,
    kf8=\EOj, kf9=\ENo, khome=\E[H, kind=\E[S, kri=\E[T,
    ll=\E[24H, mc4=\E[?4i, mc5=\E[?5i, nel=\EE,
    pfx=\E[%p1%d;%p2%l%02dq\s\s\sF%p1%ld\s\s\s\s\s
\s\s\s\s\s\s%p2%s,
    pln=\E[%p1%d;0;0;0q%p2%:-16.16s, rc=\E8, rev=\E[7m,
    ri=\EM, rmacs=^O, rmir=\E[4l, rmln=\E[2p, rmso=\E[m,
    rmul=\E[m, rs2=\Ec\E[?3l, sc=\E7,
    sgr=\E[0%?%p6%t;1%;%?%p5%t;2%;%?%p2%t;4%;%?%p4%t;5%;
%?%p3%p1% | %t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,
    sgr0=\E[m^O, smacs=^N, smir=\E[4h, smln=\E[p,
    smso=\E[7m, smul=\E[4m, tsl=\E7\E[25;%i%p1%dx,
```

Types of Capabilities in the Sample Entry

The example shows the formats for the three types of *terminfo* capabilities: Boolean, numeric, and string. All capabilities specified in the *terminfo* source file must be followed by commas, including the last capability in the source file.

Capabilities are referenced by their capability names (as shown in the previous tables). Boolean capabilities are specified simply by their comma separated cap names.

Numeric capabilities are followed by the character '#' and then a positive integer value. Thus, in the sample, **cols** (which shows the number of columns available on a device) is assigned the value **80** for the AT&T 610. (Values for numeric capabilities may be specified in decimal, octal or hexadecimal, using normal C conventions.)

Finally, string-valued capabilities such as **el** (clear to end of line sequence) are listed by a two- to five-character cap name, an '=', and a string ended by the next comma. A delay in milliseconds may appear anywhere in such a capability, enclosed in **$<..>** brackets, as in **el=\EK$<3>**. Padding characters are supplied by **tputs**( ). The delay can be any of the following: a number (**5**), a number followed by a '*' (**5***), a number followed by a '/' (**5/**), or a number followed by both (**5*/**). A '*' shows that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert characters, the factor is still the number of lines affected. This is always 1 unless the device has **in** and the software uses it.) When a '*' is specified, it is sometimes useful to give a delay of the form **3.5** to specify a delay per unit to tenths of milliseconds. Only one decimal place is allowed.

A slash '/' indicates that the padding is mandatory. If a device has **xon** defined, the padding information is advisory and is only used for cost estimates or when the device is in raw mode. Mandatory padding is transmitted regardless of the setting of **xon**. If padding (whether advisory or mandatory) is specified for **bel** or **flash**, however, it is always used, regardless of whether **xon** is specified.

terminfo offers notation for encoding special characters. Both **\E** and **\e** map to an ESCAPE character, **^x** maps to a control–*x* for any appropriate *x*, and the sequences **\n**, **\l**, **\r**, **\t**, **\b**, **\f**, and **\s** give a newline, linefeed, return, tab, backspace, formfeed, and space, respectively. Other escapes include: **\^** for caret (^); **** for backslash (\); **\,** for comma (,); **\:** for colon (:); and **\0** for null. (**\0** will actually produce **\200**, which does not terminate a string but behaves as a null character on most devices, providing CS7 is specified. (See **stty**(1).) Finally, characters may be given as three octal digits after a backslash, for example **\123**.

Sometimes individual capabilities must be commented out. Capabilities are defined in a left-to-right order, therefore, a prior definition overrides a later definition. To comment out a capability, put a period before the capability name.

Preparing Descriptions

The most effective way to prepare a device description is by imitating the description of a similar device in *terminfo* and building up a description gradually, testing partial descriptions with **vi**(1) to make sure that they are correct. A very unusual device may expose deficiencies in the ability of the *terminfo* file to describe it, or with the ability of **vi**(1) to work with that device. To test a new device description, set the environment variable **TERMINFO** to the pathname of a directory containing the compiled description you are working on so that programs look there rather than in */usr/lib/terminfo*. To get the padding for insert-line correct (if the device manufacturer did not document it) a severe test is to comment out **xon**, edit a large file at 9600 baud with **vi**(1), delete 16 or so lines from the middle of the screen, and then hit the **u** key

several times quickly. If the display is corrupted, more padding is usually needed. A similar test can be used for insert-character.

Section 1-1 Basic Capabilities

The number of columns on each line for a device is given by the **cols** numeric capability. If the device has a screen, the number of lines on the screen is indicated by the **lines** capability. If the device wraps around to the beginning of the next line when it reaches the right margin, it should have the **am** capability.

If the terminal can clear its screen, leaving the cursor in the home position, the **clear** string capability is used. If the terminal overstrikes (rather than clearing a position when a character is struck over) it should have the **os** capability. If the device is a printing terminal, with no soft copy unit, specify both **hc** and **os**. If there is a way to move the cursor to the left edge of the current row, specify this as **cr**. (Normally this will be carriage return, control M.) If there is a way to produce an audible signal, such as a bell or a beep, specify it as **bel**. If, like most devices, the device uses the xon-xoff flow-control protocol, specify **xon**.

If there is a way to move the cursor one position to the left, such as backspace, that capability should be given as **cubl**. Similarly, sequences to move to the right, up, and down should be given as **cufl**, **cuul**, and **cudl**, respectively. These local cursor motions must not alter the text they pass over; for example, you would not normally use "**cufl**=\s" because the space would erase the character moved over.

NOTE: A very important point here is that the local cursor motions encoded in *terminfo* are undefined at the left and top edges of a screen terminal. Programs should never attempt to backspace around the left edge, unless **bw** is specified, and should never attempt to go up locally off the top. To scroll text up, a program goes to the bottom left corner of the screen and sends the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin**. These versions have the same semantics as **ind** and **ri**, except that they take one parameter and scroll the number of lines specified by that parameter. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cufl** from the last column. Backward motion from the left edge of the screen is possible only when **bw** is specified. In this case, **cubl** moves to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the device has switch-selectable automatic margins, **am** should be specified in the *terminfo* source file. In this case, initialization strings should turn on this option, if possible. If the device has a command that moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the device has no **cr** and **lf** it may still be possible to make a working **nel** out of one or both of

them.

These capabilities suffice to describe hardcopy and screen terminals. Thus the AT&T 5320 hardcopy terminal is described as follows:

```
5320|att5320|AT&T 5320 hardcopy terminal,
    am, hc, os,
    cols#132,
    bel=^G, cr=\r, cubl=\b, cndl=\n,
    dchl=\E[P, dll=\E[M,
    ind=\n,
```

while the Lear Siegler ADM–3 is described as

```
adm3 | lsi adm3,
  am, bel=^G, clear=^Z, cols#80, cr=^M, cubl=^H,
  cudl=^J, ind=^J, lines#24,
```

Section 1-2 Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with **printf**(3S)-like escapes (**%x**) in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory-relative cursor addressing, it can be indicated by **mrcup**.

The parameter mechanism uses a stack and special % codes to manipulate the stack in the manner of Reverse Polish Notation (postfix). Typically a sequence pushes one of the parameters onto the stack and then prints it in some format. Often more complex operations are necessary. Operations are in postfix form with the operands in the usual order. That is, to subtract 5 from the first parameter, one would use **%p1%{5}%–**.

The % encodings have the following meanings:

%%	outputs '%'
%[[:]*flags*][*width*[.*precision*]][**doxXs**]	
	as in printf, flags are **[–+#]** and space
%c	print pop() gives %c
%p[1-9]	push i^{th} parm
%P[a-z]	set variable [a-z] to pop()
%g[a-z]	get variable [a-z] and push it
%'*c*'	push char constant *c*
%{*nn*}	push decimal constant *nn*
%l	push strlen(pop())
%+ %– %* %/ %m	
	arithmetic (%m is mod): push(pop integer2() op pop integer1())
%& %| %^	bit operations: push(pop integer2() op pop integer1())
%= %> %<	logical operations: push(pop() op pop())
%A %O	logical operations: and, or
%! %~	unary operations: push(op pop())
%i	(for ANSI terminals)
	add 1 to first parm, if one parm present,

several times quickly. If the display is corrupted, more padding is usually needed. A similar test can be used for insert-character.

Section 1-1 Basic Capabilities

The number of columns on each line for a device is given by the **cols** numeric capability. If the device has a screen, the number of lines on the screen is indicated by the **lines** capability. If the device wraps around to the beginning of the next line when it reaches the right margin, it should have the **am** capability.

If the terminal can clear its screen, leaving the cursor in the home position, the **clear** string capability is used. If the terminal overstrikes (rather than clearing a position when a character is struck over) it should have the **os** capability. If the device is a printing terminal, with no soft copy unit, specify both **hc** and **os**. If there is a way to move the cursor to the left edge of the current row, specify this as **cr**. (Normally this will be carriage return, control M.) If there is a way to produce an audible signal, such as a bell or a beep, specify it as **bel**. If, like most devices, the device uses the xon-xoff flow-control protocol, specify **xon**.

If there is a way to move the cursor one position to the left, such as backspace, that capability should be given as **cubl**. Similarly, sequences to move to the right, up, and down should be given as **cufl**, **cuul**, and **cudl**, respectively. These local cursor motions must not alter the text they pass over; for example, you would not normally use "**cufl**=\s" because the space would erase the character moved over.

NOTE: A very important point here is that the local cursor motions encoded in *terminfo* are undefined at the left and top edges of a screen terminal. Programs should never attempt to backspace around the left edge, unless **bw** is specified, and should never attempt to go up locally off the top. To scroll text up, a program goes to the bottom left corner of the screen and sends the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin**. These versions have the same semantics as **ind** and **ri**, except that they take one parameter and scroll the number of lines specified by that parameter. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cufl** from the last column. Backward motion from the left edge of the screen is possible only when **bw** is specified. In this case, **cubl** moves to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the device has switch-selectable automatic margins, **am** should be specified in the *terminfo* source file. In this case, initialization strings should turn on this option, if possible. If the device has a command that moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the device has no **cr** and **lf** it may still be possible to make a working **nel** out of one or both of

them.

These capabilities suffice to describe hardcopy and screen terminals. Thus the AT&T 5320 hardcopy terminal is described as follows:

```
5320|att5320|AT&T 5320 hardcopy terminal,
    am, hc, os,
    cols#132,
    bel=^G, cr=\r, cub1=\b, cnd1=\n,
    dch1=\E[P, dl1=\E[M,
    ind=\n,
```

while the Lear Siegler ADM–3 is described as

```
adm3|lsi adm3,
  am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H,
  cud1=^J, ind=^J, lines#24,
```

Section 1-2 Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with **printf**(3S)-like escapes (**%x**) in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory-relative cursor addressing, it can be indicated by **mrcup**.

The parameter mechanism uses a stack and special % codes to manipulate the stack in the manner of Reverse Polish Notation (postfix). Typically a sequence pushes one of the parameters onto the stack and then prints it in some format. Often more complex operations are necessary. Operations are in postfix form with the operands in the usual order. That is, to subtract 5 from the first parameter, one would use **%p1%{5}%–**.

The % encodings have the following meanings:

%%	outputs '%'
%[[:]*flags*][*width*[.*precision*]][**doxXs**]	as in printf, flags are **[–+#]** and space
%c	print pop() gives %c
%p[1-9]	push i^{th} parm
%P[a-z]	set variable [a-z] to pop()
%g[a-z]	get variable [a-z] and push it
%'*c*'	push char constant *c*
%{*nn*}	push decimal constant *nn*
%l	push strlen(pop())
%+ %– %* %/ %m	arithmetic (%m is mod): push(pop integer2() op pop integer1())
%& %| %^	bit operations: push(pop integer2() op pop integer1())
%= %> %<	logical operations: push(pop() op pop())
%A %O	logical operations: and, or
%! %~	unary operations: push(op pop())
%i	(for ANSI terminals) add 1 to first parm, if one parm present,

or first two parms, if more than one parm present

%? expr %t thenpart %e elsepart %;
if-then-else, %e elsepart is optional;
else-if's are possible ala Algol 68:
%? c_1 %t b_1 %e c_2 %t b_2 %e c_3 %t b_3 %e c_4 %t b_4 %e b_5%;
c_i are conditions, b_i are bodies.

If the – flag is used with %[**doxXs**], then a colon (:) must be placed between the % and the – to differentiate the flag from the binary %– operator, for example **%:–16.16s**

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent **\E&a12c03Y** padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are zero-padded as two digits. Thus its **cup** capability is **cup**=\E&a%p2%2.2dc%p1%2.2dY$<6>.

The Micro-Term ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, like this: **cup**=^T%p1%c%p2%c. Devices that use %**c** need to be able to backspace the cursor (**cubl**), and to move the cursor up one line on the screen (**cuul**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (The library routines dealing with *terminfo* set *tty* modes so that tabs are never expanded, so **\t** is safe to send. This is essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, like this: **cup**=\E=%p1%'\s'%+%c%p2%'\s'%+%c. Sending **\E=** pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

Section 1-3 Cursor Motions

If the terminal has a fast way to return the cursor to the very upper left corner of screen, this can be given as **home**. Similarly, a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuul** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. The home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. Thus, the **\EH** sequence on Hewlett-Packard terminals cannot be used for **home** without losing some of the other features on the terminal.

If the device has row or column absolute-cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cup**. If there are parameterized local motions (for example, move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the device does not have **cup**, such as the Tektronix 4025.

If the device needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals, such as the *Concept*, with more than one page of memory. If the device has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the device for cursor addressing to work properly. This is also used for the Tektronix 4025, where **smcup** sets the command character to be the one used by **terminfo**. If the **smcup** sequence will not restore the screen after an **rmcup** sequence is output (to the state prior to outputting **rmcup**), specify **nrrmc**.

Section 1-4 Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, indicate this with **el**. If the terminal can clear from the beginning of the line to the current position inclusive, leaving the cursor where it is, indicate this with **el1**. If the terminal can clear from the current position to the end of the display, indicate this with **ed**. **ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

Section 1-5 Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, indicate this with **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, indicate this with **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter, and insert or delete that many lines, can be given as **il** and **dl**.

If the terminal has a settable destructive scrolling region (like the VT100) the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is undefined after using this command. It is possible to get the effect of insert line or delete line using this command; the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert or delete line, and is often faster even on terminals with those features.

To determine whether a terminal has destructive scrolling regions or non-destructive scrolling regions, create a scrolling region in the middle of the screen, place data on the bottom line of the scrolling region, move the cursor to the top line of the scrolling region, and do a reverse index (**ri**) followed by a delete line (**dl1**) or index (**ind**). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by the **dl1** or **ind**, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify **csr** if the terminal has non-destructive scrolling regions, unless **ind**, **ri**, **indn**, **rin**, **dl**, and **dl1** all simulate destructive scrolling.

If the terminal has the ability to define a window as part of memory, which all commands affect, indicate this with the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling a full screen may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Section 1-6 Insert/Delete Character

There are two kinds of intelligent terminals with respect to insert and delete character operations which can be described using *terminfo*. The most common insert and delete character operations affect only the characters on the current line and shift characters off the end of the line. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type "**abc def**" using local cursor motions (not spaces) between the **abc** and the **def**. Then position the cursor before the **abc** and put the terminal in insert mode. If typing characters causes the rest of the line to shift and characters to fall off the end, your terminal does not distinguish between blanks and untyped positions. If the **abc** shifts over to the **def** which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for "insert null." While these are two logically separate attributes (one line versus multiline insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

terminfo can describe both terminals that have an insert mode and terminals which send a simple sequence to open a blank position on the current line. Give as **smir** the sequence to get into insert mode. Give as **rmir** the sequence to leave insert mode. Now give as **ichl** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ichl**; terminals that send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ichl**. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds padding in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an 'insert mode' and a special code to precede each inserted character, then both **smir/rmir** and **ichl** can be given, and both are used. The **ich** capability, with one parameter, *n*, inserts *n* blanks.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in **rmp**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** affects only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n*, to delete *n* characters, and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Section 1-7 Highlighting, Underlining, and Visible Bells

Your device may have one or more kinds of display attributes that allow you to highlight selected characters when they appear on the screen. The following display modes (shown with the names by which they are set) may be available: a blinking screen (**blink**), bold or extra-bright characters (**bold**), dim or half-bright characters (**dim**), blanking or invisible text (**invis**), protected text (**prot**), a reverse-video screen (**rev**), and an alternate character set (**smacs** to enter this mode and **rmacs** to exit it). (If a command is necessary before you can enter alternate character set mode, give the sequence in **enacs** (enable alternate-character-set mode). Turning on any of these modes singly may or may not turn off other modes.

sgr0 should be used to turn off all video enhancement capabilities. It should always be specified because it represents the only way to turn off some capabilities, such as **dim** or **blink**.

You should choose one display method as *standout mode* (see *curses*(3X)) and use it to highlight error messages and other kinds of text to which you want to draw attention. Choose a form of display that provides strong contrast but that is easy on the eyes. (We recommend reverse-video plus half-bright or reverse-video alone.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many spaces are left.

Sequences to begin underlining and end underlining can be specified as **smul** and **rmul** , respectively. If the device has a sequence to underline the current character and to move the cursor one space to the right (such as the Micro-Term MIME), this sequence can be specified as **uc**.

Terminals with the "magic cookie" glitch (**xmc**) deposit special "cookies" when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), indicate this as **flash**; it must not move the cursor. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. The boolean **chts** should also be given. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which

undoes the effects of either of these modes.

If your terminal generates underlined characters by using the underline character (with no special sequences needed) even though it does not otherwise overstrike characters, then you should specify the capability **ul**. For devices on which a character overstriking another leaves both characters on the screen, specify the capability **os**. If overstrikes are erasable with a blank, then this should be indicated by specifying **eo**.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking nine parameters. Each parameter is either **0** or non-zero, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need to be supported by **sgr**; only those for which corresponding separate attribute commands exist should be supported. For example, let's assume that the terminal in question needs the following escape sequences to turn on various modes.

tparm Parameter	**Attribute**	**Escape Sequence**
	none	\E[0m
p1	standout	\E[0;4;7m
p2	underline	\E[0;3m
p3	reverse	\E[0;4m
p4	blink	\E[0;5m
p5	dim	\E[0;7m
p6	bold	\E[0;3;4m
p7	invis	\E[0;8m
p8	protect	not available
p9	altcharset	^O (off) ^N(on)

Each escape sequence requires a **0** to turn off other modes before turning on its own mode. Also, as suggested above, *standout* is set up to be the combination of *reverse* and *dim*. Also, because this terminal has no *bold* mode, *bold* is set up as the combination of *reverse* and *underline*. In addition, to allow combinations, such as *underline+blink*, the sequence to use would be **\E[0;3;5m**. The terminal doesn't have *protect* mode, either, but that cannot be simulated in any way, so **p8** is ignored. The *altcharset* mode is different in that it is either **^O** or **^N**, depending on whether it is off or on. If all modes were to be turned on, the sequence would be **\E[0;3;4;5;7;8m^N**

Now look at different output sequences, for example ;3 is output when either **p2** or **p6** is true, that is, if either *underline* or *bold* modes are turned on. Writing out the above sequences, along with their dependencies, gives the following:

Sequence	When to Output	Terminfo Translation
\E[0	always	\E[0
;3	if p2 or p6	%?%p2%p6%|%t;3%;
;4	if p1 or p3 or p6	%?%p1%p3%|%p6%|%t;4%;
;5	if p4	%?%p4%t;5%;
;7	if p1 or p5	%?%p1%p5%|%t;7%;
;8	if p7	%?%p7%t;8%;
m	always	m
^N or ^O	if p9 ^N, else ^O	%?%p9%t^N%e^O%;

Putting this all together into the **sgr** sequence gives:

sgr=\E[0%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%|%t;4%;%?%p5%t;
5%;%?%p1%p5%|%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,

Remember that **sgr** and **sgr0 must always** be specified.

Section 1-8 Keypad

If the device has a keypad that transmits sequences when the keys are pressed, this information can also be specified. Note that it is not possible to handle devices where the keypad only works in local mode. (This applies, for example, to the unshifted Hewlett-Packard 2621 keys.) If the keypad can be set to transmit or not transmit, specify these sequences as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit.

The sequences sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcubl, kcufl, kcuul, kcudl,** and **khome**, respectively. If there are function keys such as f0, f1, ..., f63, the sequences they send can be specified as **kf0, kf1, ..., kf63**. If the first 11 keys have labels other than the default f0 through f10, the labels can be given as **lf0, lf1, ..., lf10**. The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all tabs), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward/down), **kri** (scroll backward/up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1, ka3, kb2, kc1,** and **kc3**. These keys are useful when the effects of a 3 by 3 directional pad are needed. Further keys are defined above in the capabilities list.

Strings to program function keys can be specified as **pfkey**, **pfloc**, and **pfx**. A string to program screen labels should be specified as **pln**. Each of these strings takes two parameters: a function key identifier and a string to program it with. **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local mode; and **pfx** causes the string to be transmitted to the computer. The capabilities **nlab**, **lw** and **lh** define the number of programmable screen labels and their width and height. If there are commands to turn the labels on and off, give them in **smln** and **rmln**. **smln** is normally output after one or more **pln** sequences to make sure that the change becomes visible.

Section 1-9 Tabs and Initialization

If the device has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control I). A **backtab** command that moves leftward to the next tab stop can be given as **cbt**. By convention, if *tty* modes show that tabs are being expanded by the computer rather than being sent to the device, programs should not use **ht** or **cbt** (even if they are present) because the user may not have the tab stops properly set. If the device has hardware tabs that are initially set every *n* spaces when the device is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by **tput init** (see *tput*(1)) to determine whether to set the mode for hardware tab expansion and whether to set the tab stops. If the device has tab stops that can be saved in nonvolatile memory, the *terminfo* description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row).

Other capabilities include: **is1**, **is2**, and **is3**, initialization strings for the device; **iprog**, the path name of a program to be run to initialize the device; and **if**, the name of a file containing long initialization strings. These strings are expected to set the device into modes consistent with the rest of the *terminfo* description. They must be sent to the device each time the user logs in and be output in the following order: run the program **iprog**; output **is1**; output **is2**; set the margins using **mgc**, **smgl** and **smgr**; set the tabs using **tbc** and **hts**; print the file **if**; and finally output **is3**. This is usually done using the **init** option of *tput*(1); see *profile*(4).

Most initialization is done with **is2**. Special device modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. Sequences that do a harder reset from a totally unknown state can be given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is1**, **is2**, **is3**, and **if**. (The method using files, **if** and **rf**, is used for a few terminals, from */usr/lib/tabset/**; however, the recommended method is to use the initialization and reset strings.) These strings are output by **tput reset**, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs1**, **rs2**, **rs3**, and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set a terminal into 80-column mode would normally be part of **is2**, but on some terminals it causes an annoying glitch on the screen and is not normally needed because the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tabs than can be described by using **tbc** and **hts**, the sequence can be placed in **is2** or **if**.

Any margin can be cleared with **mgc**. (For instructions on how to specify commands to set and clear margins, refer to *PRINTER CAPABILITIES*.)

Section 1-10 Delays

Certain capabilities control padding in the **tty**(7) driver. These are primarily needed by hard-copy terminals, and are used by **tput init** to set *tty* modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cubl**, **ff**, and **tab** can be used to set the appropriate delay bits to be set in the **tty** driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Section 1-11 Status Lines

If the terminal has an extra *status line* that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit h19's 25th line, or the 24th line of a VT100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings that go to a given column of the status line and return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The capability **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to.

If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, e.g., **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

Section 1-12 Line Graphics

If the device has a line drawing alternate character set, the mapping of glyph to character would be given in **acsc**. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

Glyph Name	vt100+ Character
arrow pointing right	+
arrow pointing left	,
arrow pointing down	.
solid square block	0
lantern symbol	I
arrow pointing up	-
diamond	`
checker board (stipple)	a
degree symbol	f
plus/minus	g
board of squares	h
lower right corner	j
upper right corner	k
upper left corner	l
lower left corner	m
plus	n
scan line 1	o
horizontal line	q
scan line 9	s
left tee (+)	t
right tee (-|)	u
bottom tee (⊥)	v
top tee (⊤)	w
vertical line	x
bullet	~

The best way to describe a new device's line graphics set is to add a third column to the above table with the characters for the new device that produce the appropriate glyph when the device is in the alternate character set mode. For example,

Glyph Name	vt100+ Char	New tty Char
upper left corner	l	R
lower left corner	m	F
upper right corner	k	T
lower right corner	j	G
horizontal line	q	,
vertical line	x	.

Now write down the characters left to right, as in "**acsc=lRmFkTjGq\,x.**".

In addition, *terminfo* allows you to define multiple character sets. See Section 2-5 for details.

Section 1-13 Color Manipulation

Let us define two methods of color manipulation: the Tektronix method and the HP method. The Tektronix method uses a set of N predefined colors (usually 8) from which a user can select *current* foreground and background colors. Thus, a terminal can support up to N colors mixed into N*N

color-pairs to be displayed on the screen at the same time. Using the HP method, the user cannot define the foreground independently of the background, or vice-versa. Instead, the user must define an entire color-pair at once. Up to M color-pairs, made from 2*M different colors, can be defined this way. Most existing color terminals belong to one of these two classes of terminals.

The numeric variables **colors** and **pairs** define the number of colors and color-pairs that can be displayed on the screen at the same time. If a terminal can change the definition of a color (for example, the Tektronix 4100 and 4200 series terminals), this should be specified with **ccc** (can change color). To change the definition of a color (Tektronix method), use **initc** (initialize color). It requires four arguments: color number (ranging from 0 to **colors**–1) and three RGB (red, green, and blue) values (ranging from 0 to 1000).

Tektronix 4100 series terminals use a type of color notation called HLS (Hue Lightness Saturation) instead of RGB color notation. For such terminals one must define a boolean variable **hls**. The last three arguments to the **initc** string would then be HLS values: H, ranging from 0 to 360; and L and S, ranging from 0 to 100.

If a terminal can change the definitions of colors, but uses a color notation different from RGB and HLS, a mapping to either RGB or HLS must be developed.

To set current foreground or background to a given color, use **setf** (set foreground) and **setb** (set background). They require one parameter: the number of the color. To initialize a color-pair (HP method), use **initp** (initialize pair). It requires seven parameters: the number of a color-pair (range=0 to **pairs**–1), and six RGB values: three for the foreground followed by three for the background. (Each of these groups of three should be in the order RGB.) When **initc** or **initp** are used, RGB or HLS arguments should be in the order "red, green, blue" or "hue, lightness, saturation"), respectively. To make a color-pair current, use **scp** (set color-pair). **scp** takes one parameter—the number of a color-pair.

Some terminals (for example, most color terminal emulators for PCs) erase areas of the screen with current background color. In such cases, **bce** (background color erase) should be defined. The variable **op** (original pair) contains a sequence for setting the foreground and the background colors to what they were at the terminal start-up time. Similarly, **oc** (original colors) contains a control sequence for setting all colors (for the Tektronix method) or color-pairs (for the HP method) to the values they had at the terminal start-up time.

Some color terminals substitute color for video attributes. Such video attributes should not be combined with colors. Information about these video attributes should be packed into the **ncv** (no color video) variable. There is a one-to-one correspondence between the nine least significant bits of that variable and the video attributes. The following table depicts this correspondence.

Attribute	Bit Position	Decimal Value
A_STANDOUT	0	1
A_UNDERLINE	1	2
A_REVERSE	2	4
A_BLINK	3	8
A_DIM	4	16
A_BOLD	5	32
A_INVIS	6	64
A_PROTECT	7	128
A_ALTCHARSET	8	256

When a particular video attribute should not be used with colors, the corresponding **ncv** bit should be set to 1; otherwise it should be set to 0. To determine the information to pack into the **ncv** variable, you must add together the decimal values corresponding to those attributes that cannot coexist with colors. For example, if the terminal uses colors to simulate reverse video (bit number 2 and decimal value 4) and bold (bit number 5 and decimal value 32), the resulting value for **ncv** will be 36 (4 + 32).

Section 1-14 Miscellaneous

If the terminal requires other than a null (zero) character as a pad, indicate this with **pad**. Only the first character of the **pad** string is used. If the terminal does not have a pad character, specify **npc**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually Crtl-L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, **tparm(repeat_char, 'x', 10)** is the same as **xxxxxxxxxx**.

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **cmdch**. A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. The following convention is supported on some operating systems: If the environment variable CC exists, all occurrences of the prototype character are replaced with the character in CC.

Terminal descriptions that do not represent a specific kind of known terminal, such as **switch, dialup, patch**, and **network**, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to **virtual** terminal descriptions for which the escape sequences are known.) If the terminal is one of those supported by the operating system virtual terminal protocol, the terminal number can be given as **vt**. A line-turn-around sequence to be transmitted before doing reads should be specified in **rfi**.

If the device uses xon/xoff handshaking for flow control, give **xon**. Padding information should still be included so that routines can make better decisions

about costs, but actual pad characters will not be transmitted. Sequences to turn on and off xon/xoff handshaking may be given in **smxon** and **rmxon**. If the characters used for handshaking are not ^S and ^Q, they may be specified with **xonc** and **xoffc**.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, indicate this with **km**. Otherwise, software assumes that the 8th bit is parity and it is usually cleared. If strings exist to turn this "meta mode" on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm**#0 indicates that the number of lines is not fixed, but that there is more memory than fits on the screen.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal is sent to the printer. A variation, **mc5p**, takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify **mc5i** (silent printer). All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Section 1-15 Special Cases

The working model used by *terminfo* fits most terminals reasonably well. However, some terminals do not completely match that model, requiring special support by *terminfo*. They may be unusual devices or, for some reason, do not have all the features of the *terminfo* model implemented.

Terminals that cannot display tilde (~) characters, such as certain Hazeltine terminals, should indicate **hz**.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the *Concept* 100, should indicate **xenl**. Those terminals whose cursor remains on the right-most column until another character has been received, rather than wrapping immediately upon receiving the right-most character, such as the VT100, should also indicate **xenl**.

If **el** is required to get rid of standout (instead of writing normal text on top of it), **xhp** should be given.

Teleray terminals whose tabs turn all characters moved over to blanks should indicate **xt** (destructive tabs). This capability is also taken to mean that it is not possible to position the cursor on top of a "magic cookie." Therefore, to erase standout mode, it is necessary, instead, to use delete and insert line.

Those Beehive Superbee terminals which do not transmit the escape or control–C characters, should specify **xsb**, indicating that the f1 key is to be used for escape and the f2 key for control–C.

Section 1-16 Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be canceled by

placing *xx@* to the left of the capability definition, where *xx* is the capability. For example, the entry

```
att4424-2|Teletype 4424 in display function group ii,
rev@, sgr@, smul@; use=att4424,
```

defines an AT&T 4424 terminal that does not have the **rev**, **sgr**, and **smul** capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one **use** capability may be given.

PART 2 PRINTER CAPABILITIES

The *terminfo* database allows you to define capabilities of printers as well as terminals. To find out what capabilities are available for printers as well as for terminals, see the two lists under "DEVICE CAPABILITIES" that list capabilities by variable and by capability name.

Section 2-1 Rounding Values

Because parameterized string capabilities work only with integer values, we recommend that *terminfo* designers create strings that expect numeric values that have been rounded. Application designers should note this and should always round values to the nearest integer before using them with a parameterized string capability.

Section 2-2 Printer Resolution

A printer's resolution is defined to be the smallest spacing of characters it can achieve. In general printers have independent resolution horizontally and vertically. Thus the vertical resolution of a printer can be determined by measuring the smallest achievable distance between consecutive printing baselines, while the horizontal resolution can be determined by measuring the smallest achievable distance between the left-most edges of consecutive printed, identical, characters.

All printers are assumed to be capable of printing with a uniform horizontal and vertical resolution. The view of printing that *terminfo* currently presents is one of printing inside a uniform matrix: all characters are printed at fixed positions relative to each "cell" in the matrix; furthermore, each cell has the same size given by the smallest horizontal and vertical step sizes dictated by the resolution. (The cell size can be changed as will be seen later.)

Many printers are capable of proportional printing, where the horizontal spacing depends on the size of the character last printed. *terminfo* does not make use of this capability, although it does provide enough capability definitions to allow an application to simulate proportional printing.

A printer must not only be able to print characters as close together as the horizontal and vertical resolutions suggest, but also of moving to a position an integral multiple of the smallest distance away from a previous position. Thus printed characters can be spaced apart a distance that is an integral multiple of the smallest distance, up to the length or width of a single page.

Some printers can have different resolutions depending on different modes. In *normal mode*, the existing *terminfo* capabilities are assumed to work on columns and lines, just like a video terminal. Thus the old **lines** capability would give the length of a page in lines, and the **cols** capability would give the width of a page in columns. In *micro mode*, many *terminfo* capabilities work on increments of lines and columns. With some printers the micro mode may

be concomitant with normal mode, so that all the capabilities work at the same time.

Section 2-3 Specifying Printer Resolution

The printing resolution of a printer is given in several ways. Each specifies the resolution as the number of smallest steps per distance:

Specification of Printer Resolution
Characteristic Number of Smallest Steps

orhi	Steps per inch horizontally
orvi	Steps per inch vertically
orc	Steps per column
orl	Steps per line

When printing in normal mode, each character printed causes movement to the next column, except in special cases described later; the distance moved is the same as the per-column resolution. Some printers cause an automatic movement to the next line when a character is printed in the rightmost position; the distance moved vertically is the same as the per-line resolution. When printing in micro mode, these distances can be different, and may be zero for some printers.

Specification of Printer Resolution
Automatic Motion after Printing

Normal Mode:	
orc	Steps moved horizontally
orl	Steps moved vertically
Micro Mode:	
mcs	Steps moved horizontally
mls	Steps moved vertically

Some printers are capable of printing wide characters. The distance moved when a wide character is printed in normal mode may be different from when a regular width character is printed. The distance moved when a wide character is printed in micro mode may also be different from when a regular character is printed in micro mode, but the differences are assumed to be related: if the distance moved for a regular character is the same whether in normal mode or micro mode (**mcs=orc**), then the distance moved for a wide character is also the same whether in normal mode or micro mode. This doesn't mean the normal character distance is necessarily the same as the wide character distance, just that the distances don't change with a change in normal to micro mode. However, if the distance moved for a regular character is different in micro mode from the distance moved in normal mode (**mcs<orc**), the micro mode distance is assumed to be the same for a wide character printed in micro mode, as the table below shows.

placing *xx@* to the left of the capability definition, where *xx* is the capability. For example, the entry

```
att4424-2|Teletype 4424 in display function group ii,
rev@, sgr@, smul@; use=att4424,
```

defines an AT&T 4424 terminal that does not have the **rev**, **sgr**, and **smul** capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one **use** capability may be given.

PART 2 PRINTER CAPABILITIES

The *terminfo* database allows you to define capabilities of printers as well as terminals. To find out what capabilities are available for printers as well as for terminals, see the two lists under "DEVICE CAPABILITIES" that list capabilities by variable and by capability name.

Section 2-1 Rounding Values

Because parameterized string capabilities work only with integer values, we recommend that *terminfo* designers create strings that expect numeric values that have been rounded. Application designers should note this and should always round values to the nearest integer before using them with a parameterized string capability.

Section 2-2 Printer Resolution

A printer's resolution is defined to be the smallest spacing of characters it can achieve. In general printers have independent resolution horizontally and vertically. Thus the vertical resolution of a printer can be determined by measuring the smallest achievable distance between consecutive printing baselines, while the horizontal resolution can be determined by measuring the smallest achievable distance between the left-most edges of consecutive printed, identical, characters.

All printers are assumed to be capable of printing with a uniform horizontal and vertical resolution. The view of printing that *terminfo* currently presents is one of printing inside a uniform matrix: all characters are printed at fixed positions relative to each "cell" in the matrix; furthermore, each cell has the same size given by the smallest horizontal and vertical step sizes dictated by the resolution. (The cell size can be changed as will be seen later.)

Many printers are capable of proportional printing, where the horizontal spacing depends on the size of the character last printed. *terminfo* does not make use of this capability, although it does provide enough capability definitions to allow an application to simulate proportional printing.

A printer must not only be able to print characters as close together as the horizontal and vertical resolutions suggest, but also of moving to a position an integral multiple of the smallest distance away from a previous position. Thus printed characters can be spaced apart a distance that is an integral multiple of the smallest distance, up to the length or width of a single page.

Some printers can have different resolutions depending on different modes. In *normal mode*, the existing *terminfo* capabilities are assumed to work on columns and lines, just like a video terminal. Thus the old **lines** capability would give the length of a page in lines, and the **cols** capability would give the width of a page in columns. In *micro mode*, many *terminfo* capabilities work on increments of lines and columns. With some printers the micro mode may

be concomitant with normal mode, so that all the capabilities work at the same time.

Section 2-3 Specifying Printer Resolution

The printing resolution of a printer is given in several ways. Each specifies the resolution as the number of smallest steps per distance:

Specification of Printer Resolution
Characteristic Number of Smallest Steps

orhi	Steps per inch horizontally
orvi	Steps per inch vertically
orc	Steps per column
orl	Steps per line

When printing in normal mode, each character printed causes movement to the next column, except in special cases described later; the distance moved is the same as the per-column resolution. Some printers cause an automatic movement to the next line when a character is printed in the rightmost position; the distance moved vertically is the same as the per-line resolution. When printing in micro mode, these distances can be different, and may be zero for some printers.

Specification of Printer Resolution
Automatic Motion after Printing

Normal Mode:	
orc	Steps moved horizontally
orl	Steps moved vertically
Micro Mode:	
mcs	Steps moved horizontally
mls	Steps moved vertically

Some printers are capable of printing wide characters. The distance moved when a wide character is printed in normal mode may be different from when a regular width character is printed. The distance moved when a wide character is printed in micro mode may also be different from when a regular character is printed in micro mode, but the differences are assumed to be related: if the distance moved for a regular character is the same whether in normal mode or micro mode (**mcs=orc**), then the distance moved for a wide character is also the same whether in normal mode or micro mode. This doesn't mean the normal character distance is necessarily the same as the wide character distance, just that the distances don't change with a change in normal to micro mode. However, if the distance moved for a regular character is different in micro mode from the distance moved in normal mode (**mcs<orc**), the micro mode distance is assumed to be the same for a wide character printed in micro mode, as the table below shows.

Specification of Printer Resolution
Automatic Motion after Printing Wide Character

*Normal Mode or Micro Mode (*mcs = orc*):*
widcs Steps moved horizontally

*Micro Mode (*mcs < orc*):*
mcs Steps moved horizontally

There may be control sequences to change the number of columns per inch (the character pitch) and to change the number of lines per inch (the line pitch). If these are used, the resolution of the printer changes, but the type of change depends on the printer, as follows:

Specification of Printer Resolution
Changing the Character/Line Pitches

cpi	Change character pitch
cpix	If set, **cpi** changes **orhi**, otherwise changes **orc**
lpi	Change line pitch
lpix	If set, **lpi** changes **orvi**, otherwise changes **orl**
chr	Change steps per column
cvr	Change steps per line

The **cpi** and **lpi** string capabilities are each used with a single argument, the pitch in columns (or characters) and lines per inch, respectively. The **chr** and **cvr** string capabilities are each used with a single argument, the number of steps per column and line, respectively.

Using any of the control sequences in these strings implies a change in some of the values of **orc**, **orhi**, **orl**, and **orvi**. Also, the distance moved when a wide character is printed, **widcs**, changes in relation to **orc**. The distance moved when a character is printed in micro mode, **mcs**, changes similarly, with one exception: if the distance is 0 or 1, then no change is assumed (see items marked with † in the following table).

Programs that use **cpi**, **lpi**, **chr**, or **cvr** should recalculate the printer resolution (and should recalculate other values —refer to *Effect of Changing Printing Resolution*).

Specification of Printer Resolution
Effects of Changing the Character/Line Pitches

Before	*After*
Using **cpi** *with* **cpix** *clear:*	
orhi '	**orhi**
orc '	$\mathbf{orc}=\frac{\mathbf{orhi}}{V_{cpi}}$
Using **cpi** *with* **cpix** *set:*	
orhi '	$\mathbf{orhi}=\mathbf{orc}\cdot V_{cpi}$
orc '	**orc**
Using **lpi** *with* **lpix** *clear:*	
orvi '	**orvi**
orl '	$\mathbf{orl}=\frac{\mathbf{orvi}}{V_{lpi}}$
Using **lpi** *with* **lpix** *set:*	
orvi '	$\mathbf{orvi}=\mathbf{orl}\cdot V_{lpi}$
orl '	**orl**
Using **chr:**	
orhi '	**orhi**
orc '	V_{chr}
Using **cvr:**	
orvi '	**orvi**
orl '	V_{cvr}
Using **cpi** *or* **chr:**	
widcs '	$\mathbf{widcs}=\mathbf{widcs}'\frac{\mathbf{orc}}{\mathbf{orc}'}$
mcs '†	$\mathbf{mcs}=\mathbf{mcs}'\frac{\mathbf{orc}}{\mathbf{orc}'}$

V_{cpi}, V_{lpi}, V_{chr}, and V_{cvr} are the arguments used with **cpi**, **lpi**, **chr**, and **cvr**, respectively. The † mark indicates the old value.

Section 2-4 Capabilities that Cause Movement

In the following descriptions, *movement* refers to the motion of the "current position." With video terminals this would be the cursor; with some printers this is the carriage position. Other printers have different equivalents. In general, the current position is where a character would be displayed if printed.

terminfo has string capabilities for control sequences that cause movement a number of full columns or lines. It also has equivalent string capabilities for control sequences that cause movement a number of smallest steps.

String Capabilities for Motion

mcub1	Move 1 step left
mcuf1	Move 1 step right
mcuu1	Move 1 step up
mcud1	Move 1 step down
mcub	Move *N* steps left
mcuf	Move *N* steps right
mcuu	Move *N* steps up
mcud	Move *N* steps down
mhpa	Move *N* steps from the left
mvpa	Move *N* steps from the top

The latter six strings are each used with a single argument, *N*.

Sometimes the motion is limited to less than the width or length of a page. Also, some printers don't accept absolute motion to the left of the current position. *terminfo* has capabilities for specifying these limits.

Limits to Motion

mjump	Limit on use of **mcub1, mcuf1, mcuu1, mcud1**
maddr	Limit on use of **mhpa, mvpa**
xhpa	If set, **hpa** and **mhpa** can't move left
xvpa	If set, **vpa** and **mvpa** can't move up

If a printer needs to be in a micro mode for the motion capabilities described above to work, there are string capabilities defined to contain the control sequence to enter and exit this mode. A boolean is available for those printers where using a carriage return causes an automatic return to normal mode.

Entering/Exiting Micro Mode

smicm	Enter micro mode
rmicm	Exit micro mode
crxm	Using **cr** exits micro mode

The movement made when a character is printed in the rightmost position varies among printers. Some make no movement, some move to the beginning of the next line, others move to the beginning of the same line. *terminfo* has boolean capabilities for describing all three cases.

What Happens After Character Printed in Rightmost Position

sam	Automatic move to beginning of same line

Some printers can be put in a mode where the normal direction of motion is reversed. This mode can be especially useful when there are no capabilities for leftward or upward motion, because those capabilities can be built from the motion reversal capability and the rightward or downward motion capabilities. It is best to leave it up to an application to build the leftward or upward capabilities, though, and not enter them in the *terminfo* database. This allows several reverse motions to be strung together without intervening wasted steps that leave and reenter reverse mode.

Entering/Exiting Reverse Modes

slm	Reverse sense of horizontal motions
rlm	Restore sense of horizontal motions
sum	Reverse sense of vertical motions
rum	Restore sense of vertical motions

While sense of horizontal motions reversed:

mcub1	Move 1 step right
mcuf1	Move 1 step left
mcub	Move *N* steps right
mcuf	Move *N* steps left
cub1	Move 1 column right
cuf1	Move 1 column left
cub	Move *N* columns right
cuf	Move *N* columns left

While sense of vertical motions reversed:

mcuu1	Move 1 step down
mcud1	Move 1 step up
mcuu	Move *N* steps down
mcud	Move *N* steps up
cuu1	Move 1 line down
cud1	Move 1 line up
cuu	Move *N* lines down
cud	Move *N* lines up

The reverse motion modes should not affect the **mvpa** and **mhpa** absolute motion capabilities. The reverse vertical motion mode should, however, also reverse the action of the line wrapping that occurs when a character is printed in the right-most position. Thus printers that have the standard *terminfo* capability **am** defined should experience motion to the beginning of the previous line when a character is printed in the right-most position under reverse vertical motion mode.

The action when any other motion capabilities are used in reverse motion modes is not defined; thus, programs must exit reverse motion modes before using other motion capabilities.

Two miscellaneous capabilities complete the list of new motion capabilities. One of these is needed for printers that move the current position to the beginning of a line when certain control characters, such as "line-feed" or "form-feed," are used. The other is used for the capability of suspending the motion that normally occurs after printing a character.

Miscellaneous Motion Strings

docr	List of control characters causing cr
zerom	Prevent auto motion after printing next single character

Margins

terminfo provides two strings for setting margins on terminals: one for the left and one for the right margin. Printers, however, have two additional margins, for the top and bottom margins of each page. Furthermore, some printers require not using motion strings to move the current position to a margin and then fixing the margin there, but require the specification of where a margin

should be regardless of the current position. Therefore *terminfo* offers six additional strings for defining margins with printers.

Setting Margins

smgl	Set left margin at current column
smgr	Set right margin at current column
smgb	Set bottom margin at current line
smgt	Set top margin at current line
smgbp	Set bottom margin at line *N*
smglp	Set left margin at column *N*
smgrp	Set right margin at column *N*
smgtp	Set top margin at line *N*

The last four strings are used with one or more arguments that give the position of the margin or margins to set. If both of **smglp** and **smgrp** are set, each is used with a single argument, *N*, that gives the column number of the left and right margin, respectively. If both of **smgtp** and **smgbp** are set, each is used to set the top and bottom margin, respectively: **smgtp** is used with a single argument, *N*, the line number of the top margin; however, **smgbp** is used with two arguments, *N* and *M*, that give the line number of the bottom margin, the first counting from the top of the page and the second counting from the bottom. This accommodates the two styles of specifying the bottom margin for different printers. When coding a *terminfo* entry for a printer that has a settable bottom margin, only the first or second parameter should be used, depending on the printer. When writing an application that uses **smgbp** to set the bottom margin, both arguments must be given.

If only one of **smglp** and **smgrp** is set, then it is used with two arguments, the column number of the left and right margins, in that order. Likewise, if only one of **smgtp** and **smgbp** is set, then it is used with two arguments that give the top and bottom margins, in that order, counting from the top of the page. Thus when coding a *terminfo* entry for a printer that requires setting both left and right or top and bottom margins simultaneously, only one of **smglp** and **smgrp** or **smgtp** and **smgbp** should be defined; the other should be left blank. When writing an application that uses these string capabilities, the pairs should be first checked to see if each in the pair is set or only one is set, and should then be used accordingly.

In counting lines or columns, line zero is the top line and column zero is the left-most column. A zero value for the second argument with **smgbp** means the bottom line of the page.

All margins can be cleared with **mgc**.

Shadows, Italics, Wide Characters, Superscripts, Subscripts

Five new sets of strings are used to describe the capabilities printers have of enhancing printed text.

Enhanced Printing

sshm	Enter shadow-printing mode
rshm	Exit shadow-printing mode
sitm	Enter italicizing mode
ritm	Exit italicizing mode
swidm	Enter wide character mode
rwidm	Exit wide character mode
ssupm	Enter superscript mode
rsupm	Exit superscript mode
supcs	List of characters available as superscripts
ssubm	Enter subscript mode
rsubm	Exit subscript mode
subcs	List of characters available as subscripts

If a printer requires the **sshm** control sequence before every character to be shadow-printed, the **rshm** string is left blank. Thus programs that find a control sequence in **sshm** but none in **rshm** should use the **sshm** control sequence before every character to be shadow-printed; otherwise, the **sshm** control sequence should be used once before the set of characters to be shadow-printed, followed by **rshm**. The same is also true of each of the **sitm/ritm**, **swidm/rwidm**, **ssupm/rsupm**, and **ssubm/rsubm** pairs.

Note that *terminfo* also has a capability for printing emboldened text (**bold**). While shadow printing and emboldened printing are similar in that they darken the text, many printers produce these two types of print in slightly different ways. Generally, emboldened printing is done by overstriking the same character one or more times. Shadow printing likewise usually involves overstriking, but with a slight movement up and/or to the side so that the character is fatter.

It is assumed that enhanced printing modes are independent modes, so that it would be possible, for instance, to shadow print italicized subscripts.

As mentioned earlier, the amount of motion automatically made after printing a wide character should be given in **widcs**.

If only a subset of the printable ASCII characters can be printed as superscripts or subscripts, they should be listed in **supcs** or **subcs** strings, respectively. If the **ssupm** or **ssubm** strings contain control sequences, but the corresponding **supcs** or **subcs** strings are empty, it is assumed that all printable ASCII characters are available as superscripts or subscripts.

Automatic motion made after printing a superscript or subscript is assumed to be the same as for regular characters. Thus, for example, printing any of the following three examples will result in equivalent motion: Bi B_i B^i

Note that the existing **msgr** boolean capability describes whether motion control sequences can be used while in "standout mode." This capability is extended to cover the enhanced printing modes added here. **msgr** should be set for those printers that accept any motion control sequences without affecting shadow, italicized, widened, superscript, or subscript printing. Conversely, if **msgr** is not set, a program should end these modes before

attempting any motion.

Section 2-5 Alternate Character Sets

In addition to allowing you to define line graphics (described in Section 1-12), *terminfo* lets you define alternate character sets. The following capabilities cover printers and terminals with multiple selectable or definable character sets.

Alternate Character Sets

scs	Select character set *N*
scsd	Start definition of character set *N*, *M* characters
defc	Define character *A*, *B* dots wide, descender *D*
rcsd	End definition of character set *N*
csnm	List of character set names
daisy	Printer has manually changed print-wheels

The **scs**, **rcsd**, and **csnm** strings are used with a single argument, *N*, a number from 0 to 63 that identifies the character set. The **scsd** string is also used with the argument *N* and another, *M*, that gives the number of characters in the set. The **defc** string is used with three arguments: *A* gives the ASCII code representation for the character, *B* gives the width of the character in dots, and *D* is zero or one depending on whether the character is a "descender" or not. The **defc** string is also followed by a string of "image-data" bytes that describe how the character looks (see below).

Character set 0 is the default character set present after the printer has been initialized. Not every printer has 64 character sets, of course; using **scs** with an argument that doesn't select an available character set should cause a null result from *tparm*( ).

If a character set has to be defined before it can be used, the **scsd** control sequence is to be used before defining the character set, and the **rcsd** is to be used after. They should also cause a null result from *tparm ()* when used with an argument *N* that doesn't apply. If a character set still has to be selected after being defined, the **scs** control sequence should follow the **rcsd** control sequence. By examining the results of using each of the **scs**, **scsd**, and **rcsd** strings with a character set number in a call to *tparm*( ), a program can determine which of the three are needed.

Between use of the **scsd** and **rcsd** strings, the **defc** string should be used to define each character. To print any character on printers covered by *terminfo*, the ASCII code is sent to the printer. This is true for characters in an alternate set as well as normal characters. Thus the definition of a character includes the ASCII code that represents it. In addition, the width of the character in dots is given, along with an indication of whether the character should descend below the print line (such as the lower case letter *g* in most character sets). The width of the character in dots also indicates the number of image-data bytes that will follow the **defc** string. These image-data bytes indicate where in a dot-matrix pattern ink should be applied to draw the character; the number of these bytes and their form are defined below under *Dot-Mapped Graphics*.

It's easiest for the creator of *terminfo* entries to refer to each character set by number; however, these numbers are meaningless to the application developer. The **csnm** string alleviates this problem by providing names for each number.

When used with a character set number in a call to *tparm*(), the **csnm** string produces the equivalent name. These names should be used as a reference only. No naming convention is implied, although anyone who creates a *terminfo* entry for a printer should use names consistent with the names found in user documents for the printer. Application developers should allow a user to specify a character set by number (leaving it up to the user to examine the **csnm** string to determine the correct number), or by name, where the application examines the **csnm** string to determine the corresponding character set number.

These capabilities are likely to be used only with dot-matrix printers. If they are not available, the strings should not be defined. For printers that have manually changed print-wheels or font cartridges, the boolean **daisy** is set.

Section 2-6 Dot-Matrix Graphics

Dot-matrix printers typically have the capability of reproducing raster-graphics images. Three new numeric capabilities and three new string capabilities can help a program draw raster-graphics images independent of the type of dot-matrix printer or the number of pins or dots the printer can handle at one time.

Dot-Matrix Graphics

npins	Number of pins, *N*, in print-head
spinv	Spacing of pins vertically in pins per inch
spinh	Spacing of dots horizontally in dots per inch
porder	Matches software bits to print-head pins
sbim	Start printing bit image graphics, *B* bits wide
rbim	End printing bit image graphics

The **sbim** sring is used with a single argument, *B*, the width of the image in dots.

The model of dot-matrix or raster-graphics that *terminfo* presents is similar to the technique used for most dot-matrix printers: each pass of the printer's print-head is assumed to produce a dot-matrix that is *N* dots high and *B* dots wide. This is typically a wide, squat, rectangle of dots. The height of this rectangle in dots varies from one printer to the next; this is given in the **npins** numeric capability. The size of the rectangle in fractions of an inch also varies; it can be deduced from the **spinv** and **spinh** numeric capabilities. With these three values an application can divide a complete raster-graphics image into several horizontal strips, perhaps interpolating to account for different dot spacing vertically and horizontally.

The **sbim** and **rbim** strings are used to start and end a dot-matrix image, respectively. The **sbim** string is used with a single argument that gives the width of the dot-matrix in dots. A sequence of "image-data bytes" are sent to the printer after the **sbim** string and before the **rbim** string. The number of bytes is a integral multiple of the width of the dot-matrix; the multiple and the form of each byte is determined by the **porder** string as described below.

The **porder** string is a comma separated list of pin numbers optionally followed by an numerical offset. The offset, if given, is separated from the list with a semicolon. The position of each pin number in the list corresponds to a bit in an 8-bit data byte. The pins are numbered consecutively from 1 to **npins**, with 1 being the top pin. Note that the term *pin* is used loosely here; ink-jet dot-matrix printers don't have pins, but can be considered to have an equivalent method of applying a single dot of ink to paper. The bit positions in **porder** are in groups of 8, with the first position in each group the most significant bit and the last position the least significant bit. An application produces 8-bit bytes in the order of the groups in **porder**.

An application computes the *image-data bytes* from the internal image, mapping vertical dot positions in each print-head pass into 8-bit bytes, using a 1 bit where ink should be applied and 0 where no ink should be applied. This can be reversed (0 bit for ink, 1 bit for no ink) by giving a negative pin number. If a position is skipped in **porder**, a 0 bit is used. If a position has a lower case *x* instead of a pin number, a 1 bit is used in the skipped position. For consistency, a lower case *o* can be used to represent a 0 filled, skipped bit. There must be a multiple of 8 bit positions used or skipped in **porder**; if not, 0 bits are used to fill the last byte in the least significant bits. The offset, if given, is added to each data byte; the offset can be negative.

Some examples may help clarify the use of the **porder** string. The AT&T 470, AT&T 475 and C.Itoh 8510 printers provide eight pins for graphics. The pins are identified top to bottom by the 8 bits in a byte, from least significant to most. The **porder** strings for these printers would be `8,7,6,5,4,3,2,1`. The AT&T 478 and AT&T 479 printers also provide eight pins for graphics. However, the pins are identified in the reverse order. The **porder** strings for these printers would be `1,2,3,4,5,6,7,8`. The AT&T 5310, AT&T 5320, DEC LA100, and DEC LN03 printers provide six pins for graphics. The pins are identified top to bottom by the decimal values 1, 2, 4, 8, 16 and 32. These correspond to the low six bits in an 8-bit byte, although the decimal values are further offset by the value 63. The **porder** string for these printers would be `,,6,5,4,3,2,1;63`, or alternately `o,o,6,5,4,3,2,1;63`.

Section 2-7 Effect of Changing Printing Resolution

If the control sequences to change the character pitch or the line pitch are used, the pin or dot spacing may change:

Dot-Matrix Graphics
Changing the Character/Line Pitches

cpi	Change character pitch
cpix	If set, **cpi** changes **spinh**
lpi	Change line pitch
lpix	If set, **lpi** changes **spinv**

Programs that use **cpi** or **lpi** should recalculate the dot spacing:

Dot-Matrix Graphics
Effects of Changing the Character/Line Pitches

Before	*After*
Using **cpi** *with* **cpix** *clear:*	
spinh '	**spinh**
Using **cpi** *with* **cpix** *set:*	
spinh '	$\textbf{spinh=spinh'}\cdot\frac{\textbf{orhi}}{\textbf{orhi'}}$
Using **lpi** *with* **lpix** *clear:*	
spinv '	**spinv**
Using **lpi** *with* **lpix** *set:*	
spinv '	$\textbf{spinv=spinv'}\cdot\frac{\textbf{orhi}}{\textbf{orhi'}}$
Using **chr:**	
spinh '	**spinh**
Using **cvr:**	
spinv '	**spinv**

orhi' and **orhi** are the values of the horizontal resolution in steps per inch, before using **cpi** and after using **cpi**, respectively. Likewise, **orvi'** and **orvi** are the values of the vertical resolution in steps per inch, before using **lpi** and after using **lpi**, respectively. Thus, the changes in the dots per inch for dot-matrix graphics follow the changes in steps per inch for printer resolution.

Section 2-8 Print Quality

Many dot-matrix printers can alter the dot spacing of printed text to produce near letter-quality printing or draft-quality printing. Usually it is important to be able to choose one or the other because the rate of printing generally falls off as the quality improves. There are three new strings used to describe these capabilities.

Print Quality

snlq	Set near-letter quality print
snrmq	Set normal quality print
sdrfq	Set draft quality print

The capabilities are listed in decreasing levels of quality. If a printer doesn't have all three levels, one or two of the strings should be left blank as appropriate.

Section 2-9 Printing Rate and Buffer Size

Because there is no standard protocol that can be used to keep a program synchronized with a printer, and because modern printers can buffer data before printing it, a program generally cannot determine at any time what has been printed. Two new numeric capabilities can help a program estimate what has been printed.

Print Rate/Buffer Size

cps	Nominal print rate in characters per second
bufsz	Buffer capacity in characters

cps is the nominal or average rate at which the printer prints characters; if this value is not given, the rate should be estimated at one-tenth the prevailing baud rate. **bufsz** is the maximum number of subsequent characters buffered before the guaranteed printing of an earlier character, assuming proper flow control has been used. If this value is not given it is assumed that the printer does not buffer characters, but prints them as they are received.

As an example, if a printer has a 1000-character buffer, then sending the letter *a* followed by 1000 additional characters is guaranteed to cause the letter *a* to print. If the same printer prints at the rate of 100 characters per second, then it should take 10 seconds to print all the characters in the buffer, less if the buffer is not full. By keeping track of the characters sent to a printer, and knowing the print rate and buffer size, a program can synchronize itself with the printer.

Note that most printer manufacturers advertise the maximum print rate, not the nominal print rate. A good way to get a value to put in for **cps** is to generate a few pages of text, count the number of printable characters, and then see how long it takes to print the text.

Applications that use these values should recognize the variability in the print rate. Straight text, in short lines, with no embedded control sequences will probably print at close to the advertised print rate and probably faster than the rate in **cps**. Graphics data with a lot of control sequences, or very long lines of text, will print at well below the advertised rate and below the rate in **cps**. If the application is using **cps** to decide how long it should take a printer to print a block of text, the application should pad the estimate. If the application is using **cps** to decide how much text has already been printed, it should shrink the estimate. The application thus errs in favor of the user, who wants, above all, to see all the output in its correct place.

FILES

/usr/lib/terminfo/?/*	Compiled terminal description database
/usr/lib/.COREterm/?/*	Subset of compiled terminal description database
/usr/lib/tabset/*	Tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs)

SEE ALSO

tput(1), captoinfo(1M), infocmp(1M), tic(1M), printf(3S), curses(3X), term(5), tty(7)

WARNINGS

As described in the *Tabs and Initialization* section above, a terminal's initialization strings, **is1**, **is2**, and **is3**, if defined, must be output before a *curses*(3X) program is run. An available mechanism for outputting such strings is **tput init** (see *tput*(1) and *profile*(4)).

If a null character (\0) is encountered in a string, the null and all characters after it are lost. Therefore it is not possible to code a null character (\0) and send it to a device (either terminal or printer). The suggestion of sending a \0200, where a \0 (null) is needed can succeed only if the device (terminal or printer) ignores the eighth bit. For example, because all eight bits are used in the standard international ASCII character set, devices that adhere to this standard will treat \0200 differently from \0.

Tampering with entries in */usr/lib/.COREterm/?/** or */usr/lib/terminfo/?/** (for example, changing or removing an entry) can affect programs such as *vi*(1) that expect the entry to be present and correct. In particular, removing the description for the dumb terminal causes unexpected problems.

NAME

timezone – set default system time zone

SYNOPSIS

/etc/TIMEZONE

DESCRIPTION

This file sets and exports the time zone environment variable **TZ**.

This file is used in other files that must know the time zone.

The syntax of **TZ** is as follows:

TZ	→	*zone* | *zone signed_time* | *zone signed_time zone* | *zone signed_time zone dst*
zone	→	*letter letter letter*
signed_time	→	*sign time* | *time*
time	→	*hour* | *hour : minute* | *hour : minute : second*
dst	→	*signed_time* | *signed_time ; dst_date , dst_date* | *; dst_date , dst_date*
dst_date	→	*date / time*
date	→	J*julianone* | *julianzero* | M*month.week.day*
month	→	*1* | *2* | .. | *12*
week	→	*1* | *2* | *3* | *4* | *5*
day	→	*0* | *1* | *2* | *3* | *4* | *5* | *6*
letter	→	*a* | *A* | *b* | *B* | ... | *z* | *Z*
hour	→	*00* | *01* | ... | *23*
minute	→	*00* | *01* | ... | *59*
second	→	*00* | *01* | ... | *59*
julianzero	→	*0* | *1* | ...| *365*
julianone	→	*1* | *2* | ...| *366*
sign	→	– | +

EXAMPLES

A simple */etc/TIMEZONE* file might look like the following:

```
#       Time Zone
TZ=EST5EDT
export TZ
```

A simple setting for New Jersey could be:

```
TZ=EST5EDT
```

where EST is the abbreviation for the main time zone, 5 is the difference, in hours, between (UTC) (Universal Time Code, formerly Greenwich Mean Time) and the main time zone, and EDT is the abbreviation for the alternate time zone.

The most complex representation of the same setting, for the year 1986, is as follows:

```
TZ="EST5:00:00EDT4:00:00;117/2:00:00,299/2:00:00"
```

EST is the abbreviation for the main time zone, 5:00:00 is the difference, in hours, minutes, and seconds between UTC and the main time zone, EDT is the abbreviation for the alternate time zone, 4:00:00 is the difference, in hours, minutes, and seconds between UTC and the alternate time zone, 117 is the number of the day of the year (Julian day 0-365) when the alternate time zone will take effect, 2:00:00 is the number of hours, minutes, and seconds past midnight when the alternate time zone will take effect, 299 is the number of the day of the year when the alternate time zone will end, and 2:00:00 is the number of hours, minutes, and seconds past midnight when the alternate time zone will end.

A southern hemisphere setting such as the Cook Islands could be as follows:

```
TZ="KDT9:30KST10:00;64/5:00,303/20:00"
```

This setting means that KDT is the abbreviation for the main time zone, KST is the abbreviation for the alternate time zone, KST is 9 hours and 30 minutes later than UTC, KDT is 10 hours later than UTC, the starting date of KDT is the 64th day at 5 AM, and the ending date of KDT is the 303rd day at 8 PM.

This TZ

```
TZ="EST5:00:00EDT4:00:00;J117/2:00:00,J299/2:00:00"
```

is exactly the same as the one given above, except the day of the year when the alternate timezone begins and ends is based on a different Julian number scheme. The Julian numbers have the range 1-365, and leap years are not taken into account. February 28 is day 59 and March 1 is day 60. It is impossible to explicitly refer to the occasional February 29.

This TZ

```
TZ="EST5:00:00EDT4:00:00;M3.2.0/2:00:00,M10.3.1/2:00:00"
```

uses the third and final scheme of specifying the times for the alternative timezone beginning and ending. The beginning and ending days are specified as a floating day (Ex. The second Tuesday of March). The M says to use the floating scheme, followed by the month, week, and day. The example above gives the beginning day as the 2nd Sunday in March, and the ending day as the third Monday in October. A week value of 5 says to use the last week that contains the proper day. I.E. The last Tuesday of November would be M11.5.2

Starting and ending times are relative to the alternate time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year are used and the time is 2 AM. If the start and end dates are provided but the time is not provided, the time is midnight.

Note that in most installations, **TZ** is set to the correct value by default when the user logs on, by the local */etc/profile* file (see **profile**(4)).

NOTES

When the longer format is used, **TZ** variable must be surrounded by double quotes as shown.

The system administrator must change the Julian start and end days annually if the longer form of the TZ variable is used.

Setting the time during the interval of change from the main time zone to the alternate time zone or vice versa can produce unpredictable results.

SEE ALSO

rc2(1M), ctime(3C), profile(4), environ(5)

NAME

unistd – file header for symbolic constants

SYNOPSIS

#include <unistd.h>

DESCRIPTION

The header file *<unistd.h>* lists the symbolic constants and structures not already defined or declared in some other header file, as follows:

```
/* Symbolic constants for the "access" routine: */

#define R_OK        4          /*Test for Read permission */
#define W_OK        2          /*Test for Write permission */
#define X_OK        1          /*Test for eXecute permission */
#define F_OK        0          /*Test for existence of File */

#define F_ULOCK     0          /*Unlock a previously locked region */
#define F_LOCK      1          /*Lock a region for exclusive use */
#define F_TLOCK     2          /*Test and lock a region for exclusive use */
#define F_TEST      3          /*Test a region for other processes locks */

/*Symbolic constants for the "lseek" routine: */

#define SEEK_SET    0          /* Set file pointer to "offset" */
#define SEEK_CUR    1          /* Set file pointer to current plus "offset" */
#define SEEK_END    2          /* Set file pointer to EOF plus "offset" */

/*Pathnames:*/

#define GF_PATH     /etc/group /*Pathname of the group file */
#define PF_PATH     /etc/passwd/*Pathname of the passwd file */
```

NAME

utmp, wtmp – utmp and wtmp entry formats

SYNOPSIS

#include <sys/types.h>
#include <utmp.h>

DESCRIPTION

These files, which hold user and accounting information for such commands as **who**(1), **write**(1), and **login**(1), have the following structure as defined by *<utmp.h>*:

```
#define  UTMP_FILE      "/etc/utmp"
#define  WTMP_FILE      "/etc/wtmp"
#define  ut_name        ut_user


struct utmp {
        char        ut_user[8];         /* User login name */
        char        ut_id[4];           /*/ etc/inittab ID (usually line #)*/
        char        ut_line[12];        /* Device name (console, lnxx) */
        short       ut_pid;             /* Process ID */
        short       ut_type;            /* Type of entry */
        struct      exit_status {
          short       e_termination;    /* Proc termination status */
          short       e_exit;           /* Process exit status */
        } ut_exit;                      /* The exit status of a process
                                         * marked as DEAD_PROCESS. */
        time_t      ut_time;            /* time entry was made */
};

/*  Definitions for ut_type  */
#define  EMPTY          0
#define  RUN_LVL        1
#define  BOOT_TIME      2
#define  OLD_TIME       3
#define  NEW_TIME       4
#define  INIT_PROCESS   5           /* Process spawned by "init" */
#define  LOGIN_PROCESS  6           /* "getty" process waiting for login */
#define  USER_PROCESS   7           /* User process */
#define  DEAD_PROCESS   8
#define  ACCOUNTING     9
#define  UTMAXTYPE      ACCOUNTING  /* Largest legal value of ut_type */
```

```
/*  Special strings or formats used in the "ut_line" field when  */
/*  accounting for something.other than a process  */
/*  No string for the ut_line field can be more than 11 chars +  */
/*  a NULL in length  */
#define  RUNLVL_MSG  "run-level %c"
#define  BOOT_MSG    "system boot"
#define  OTIME_MSG   "old time"
#define  NTIME_MSG   "new time"
```

FILES

/etc/utmp
/etc/wtmp

SEE ALSO

login(1), who(1), write(1), getut(3C)

NAME

uucp – networking support files

DESCRIPTION

The */usr/lib/uucp* directory contains the Basic Networking Utilities support files. Changes to these files can be made using menus or by editing them manually. The support files are as follows:

Devices Contains information concerning the location and line speed of the automatic call unit, direct links, and network devices.

Dialers Contains character strings required to negotiate with network devices (automatic calling devices) in the establishment of connections to remote computers (non 801-type dialers).

Systems Contains information needed by the **uucico** daemon and the **cu** program to establish a link to a remote computer. It contains information such as the name of the remote computer, the name of the connecting device associated with the remote computer, when the computer can be reached, telephone number, login ID, and password.

Dialcodes Contains dial-code abbreviations that may be used in the phone number field of **Systems** file entries.

Permissions Defines the level of access that is granted to computers when they attempt to transfer files or remotely execute commands on your computer.

Poll Defines computer that are to be polled by your system and when they are polled.

Sysfiles Assigns different or multiple files to be used by **uucico** and **cu** as **Systems**, **Devices**, and **Dialers**.

SEE ALSO

rc(4)

NAME

intro – introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, and so forth.

NAME

ascii – map of ASCII character set

DESCRIPTION

ascii is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. It contains:

000	nul	001	soh	002	stx	003	etx	004	eot	005	enq	006	ack	007	bel
010	bs	011	ht	012	nl	013	vt	014	np	015	cr	016	so	017	si
020	dle	021	dc1	022	dc2	023	dc3	024	dc4	025	nak	026	syn	027	etb
030	can	031	em	032	sub	033	esc	034	fs	035	gs	036	rs	037	us
040	sp	041	!	042	"	043	#	044	$	045	%	046	&	047	´
050	(| 051 |)	052	*	053	+	054	,	055	–	056	.	057	/
060	0	061	1	062	2	063	3	064	4	065	5	066	6	067	7
070	8	071	9	072	:	073	;	074	<	075	=	076	>	077	?
100	@	101	A	102	B	103	C	104	D	105	E	106	F	107	G
110	H	111	I	112	J	113	K	114	L	115	M	116	N	117	O
120	P	121	Q	122	R	123	S	124	T	125	U	126	V	127	W
130	X	131	Y	132	Z	133	[	134	\	135	]	136	^	137	_
140	`	141	a	142	b	143	c	144	d	145	e	146	f	147	g
150	h	151	i	152	j	153	k	154	l	155	m	156	n	157	o
160	p	161	q	162	r	163	s	164	t	165	u	166	v	167	w
170	x	171	y	172	z	173	{	174	|	175	}	176	~	177	del

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0a	nl	0b	vt	0c	np	0d	cr	0e	so	0f	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1a	sub	1b	esc	1c	fs	1d	gs	1e	rs	1f	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	´
28	(| 29 |)	2a	*	2b	+	2c	,	2d	–	2e	.	2f	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3a	:	3b	;	3c	<	3d	=	3e	>	3f	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4a	J	4b	K	4c	L	4d	M	4e	N	4f	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5a	Z	5b	[	5c	\	5d	]	5e	^	5f	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6a	j	6b	k	6c	l	6d	m	6e	n	6f	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7a	z	7b	{	7c	|	7d	}	7e	~	7f	del

NAME

environ – user environment

DESCRIPTION

An array of strings called the *environment* is made available by **exec**(2) when a process begins. By convention, these strings have the form *name=value*. The following names, also called environment variables or shell variables, are used by various commands:

CFTIME The default format string to be used by the **date**(1) command and the **ascftime**( ) and **cftime**( ) routines (see **ctime**(3C)). If **CFTIME** is not set or is **NULL**, the default format string specified in the */lib/cftime/LANGUAGE* file (if it exists) is used in its place (see **cftime**(4)).

CHRCLASS A value that corresponds to a file in */lib/chrclass* containing character classification and conversion information. This information is used by commands (such as **cat**(1), **ed**(1), or **sort**(1)), to classify characters as alphabetic, printable, uppercase, and so forth, and to convert characters to uppercase or lowercase.

When a program or command begins execution, the tables containing this information are initialized based on the value of **CHRCLASS**. If **CHRCLASS** is non-existent, null, set to a value for which no file exists in */lib/chrclass*, or errors occur while reading the file, the ASCII character set is used. During execution, a program or command can change the values in these tables by calling the **setchrclass**() routine. For more detail, see **ctype**(3C).

These tables are created using the **chrtbl**(1M) command.

HOME The name of the user's login directory, set by **login**(1) from the password file (see **passwd**(4)).

LANGUAGE A language for which a printable file by that name exists in */lib/cftime*. This information is used by commands (such as **date**(1), **ls**(1), and **sort**(1)), to print date and time information in the language specified.

If **LANGUAGE** is non-existent, **NULL**, set to a value for which no file exists in */lib/cftime*, or errors occur while reading the file, the last language requested will be used. (If no language has been requested, the language **usa_english** is assumed.) For a description of the content of files in */lib/cftime*, refer to **cftime**(4).

PATH The sequence of directory prefixes that **sh**(1), **time**(1), **nice**(1), **nohup**(1), and so forth, apply in searching for a file known by an incomplete path name. The prefixes are separated by colons. **login**(1) sets `PATH=:/bin:/usr/bin`(see **sh**(1)).

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as **mm**(1) or **vi**(1), which may exploit special capabilities of that terminal.

TZ Time zone information. The simplest format is **xxx***n***zzz** where **xxx** is the standard local time zone abbreviation, **n** is the difference in hours from UTC (Universal Time Code, formerly Greenwich Mean Time), and **zzz** is the abbreviation for an alternate time zone (usually the daylight-saving local time zone), if any; for example,

```
TZ="EST5EDT"
```

The most complex format allows you to specify the difference in hours of the alternate time zone from UTC and the starting day and time and ending day and time for using this alternate time zone. For example, in 1985 the complex format corresponding to the simple example above is as follows:

```
TZ="EST5:00:00EDT4:00:00;118/2:00:00,300/2:00:00"
```

When the complex format is used, it must be surrounded by double quotes (see **ctime**(3C) and **timezone**(4)).

Further names may be placed in the environment by the *export* command and *name=value* arguments in **sh**(1), or by **exec**(2). It is unwise to conflict with certain shell variables that are frequently exported by **.profile** files: **MAIL, PS1, PS2, IFS** (see **profile**(4)).

NOTES

Administrators should note the following: if you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight time is starting or ending), and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

SEE ALSO

mm (1), cat(1), date(1), ed(1), env(1), ls(1), login(1), nice(1), nohup(1), sh(1), sort(1), time(1), vi(1), chrtbl(1M), cftime(4), passwd(4), profile(4), timezone(4), exec(2), ctime(3C), ctype(3C)

NAME

fcntl – file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

The **fcntl**(2) function provides for control over open files. This include file describes **requests** and **arguments** to **fcntl** and **open**(2).

```
/* Flag values accessible to open(2) and fcntl(2) */
/* (The first three can only be set by open) */
#define O_RDONLY    0
#define O_WRONLY    1
#define O_RDWR      2
#define O_NDELAY    04      /*Non-blocking I/O */
#define O_APPEND    010     /*append(writes guaranteed at end) */
#define O_SYNC      020     /* synchronous write option */

/* Flag values accessible only to open(2) */
#define O_CREAT     00400   /*open with file create(uses 3rd open arg)*/
#define O_TRUNC     01000   /*open with truncation */
#define O_EXCL      02000   /*exclusive open */

/* fcntl(2) requests */
#define F_DUPFD     0       /*Duplicate fildes */
#define F_GETFD     1       /*Get fildes flags */
#define F_SETFD     2       /*Set fildes flags */
#define F_GETFL     3       /*Get file flags */
#define F_SETFL     4       /*Set file flags */
#define F_GETLK     5       /*Get file lock */
#define F_SETLK     6       /*Set file lock */
#define F_SETLKW    7       /*Set file lock and wait */

/* file segment locking control structure */
struct flock {
      short l_type;
      short l_whence;
      long  l_start;
      long  l_len;      /*if 0 then until EOF */
      short l_sysid;    /*returned with F_GETLK*/
      short l_pid;      /*returned with F_GETLK*/
}

/* file segment locking types */
#define F_RDLCK 01    /*Read lock */
#define F_WRLCK 02    /*Write lock */
#define F_UNLCK 03    /*Remove locks */
```

SEE ALSO

fcntl(2), open(2)

NAME

math – math functions and constants

SYNOPSIS

#include <math.h>

DESCRIPTION

This file contains declarations of all the functions in the Math Library (described in Section 3M), as well as various functions in the C Library (Section 3C) that return floating-point values.

It defines the structure and constants used by the **matherr**(3M) error-handling mechanisms, including the following constant used as an error-return value:

HUGE	The maximum value of a single-precision floating-point number.

The following mathematical constants are defined for user convenience:

M_E	The base of natural logarithms (e).
M_LOG2E	The base-2 logarithm of e.
M_LOG10E	The base-10 logarithm of e.
M_LN2	The natural logarithm of 2.
M_LN10	The natural logarithm of 10.
M_PI	π, the ratio of the circumference of a circle to its diameter.
M_PI_2	$\pi/2$.
M_PI_4	$\pi/4$.
M_1_PI	$1/\pi$.
M_2_PI	$2/\pi$.
M_2_SQRTPI	$2/\sqrt{\pi}$.
M_SQRT2	The positive square root of 2.
M_SQRT1_2	The positive square root of 1/2.

For the definitions of various machine-dependent constants, see the description of the **<values.h>** header file.

SEE ALSO

intro(3), matherr(3M), values(5)

NAME

prof – profile within a function

SYNOPSIS

```
#define MARK
#include <prof.h>

void MARK (name)
```

DESCRIPTION

MARK introduces a mark called **name** that is treated the same as a function entry point. Execution of the mark adds to a counter for that mark, and program-counter time spent is accounted to the immediately preceding mark or to the function if there are no preceding marks within the active function.

name may be any combination of numbers or underscores. Each **name** in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol MARK must be defined before the header file *<prof.h>* is included. This may be defined by a preprocessor directive as in the synopsis, or by a command line argument, i.e:

```
cc -p -DMARK foo.c
```

If MARK is not defined, the **MARK** (name) statements may be left in the source files containing them and are ignored.

EXAMPLES

In this example, marks can be used to determine how much time is spent in each loop. Unless this example is compiled with **MARK** defined on the command line, the marks are ignored.

```
#include <prof.h>
foo( )
{
        int i, j;
        .
        .
        .
        MARK(loop1);
        for (i = 0; i < 2000; i++) {
                . . .
        }
        MARK(loop2);
        for (j = 0; j < 2000; j++) {
                . . .
        }
}
```

SEE ALSO

prof(1), profil(2), monitor(3C)

NAME

regexp – regular expression compile and match routines

SYNOPSIS

```
#define INIT <declarations>
#define GETC() <getc code>
#define PEEKC() <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>

#include <regexp.h>

char *compile (instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;
int eof;

int step (string, expbuf)
char *string, *expbuf;

extern char *loc1, *loc2, *locs;

extern int circf, sed, nbra;
```

DESCRIPTION

This page describes general-purpose regular expression matching routines in the form of **ed**(1), defined in *<regexp.h>*. Programs such as **ed**(1), **sed**(1), **grep**(1), **bs**(1), and **expr**(1), which perform regular expression matching, use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is complex. Programs that include this file must have the following five macros declared before the `#include <regexp.h>` statement. These macros are used by the **compile** routine:

GETC()	Return the value of the next character in the regular expression pattern. Successive calls to **GETC()** should return successive characters of the regular expression.
PEEKC()	Return the next character in the regular expression. Successive calls to **PEEKC()** should return the same character [which should also be the next character returned by **GETC()**].
UNGETC(c)	Cause the argument **c** to be returned by the next call to **GETC()** and PEEKC(). No more that one character of pushback is ever needed and this character is guaranteed to be the last character read by **GETC()**. The value of the macro **UNGETC(c)** is always ignored.
RETURN(pointer)	This macro is used on normal exit of the **compile** routine. The value of the argument **pointer** is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.
ERROR(val)	This is the abnormal return from the **compile** routine. The argument **val** is an error number as listed in the following table. This call should never return.

ERROR	MEANING
11	Range endpoint too large.
16	Bad number.
25	"\digit" out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	\(\) imbalance.
43	Too many \(.
44	More than 2 numbers given in \{ \}.
45	} expected after \.
46	First number exceeds second in \{ \}.
49	[ ] imbalance.
50	Regular expression overflow.

The syntax of the **compile** routine is as follows:

```
compile(instring, expbuf, endbuf, eof)
```

The first parameter **instring** is never used explicitly by the **compile** routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the **INIT** declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((**char** *) **0**) for this parameter.

The next parameter **expbuf** is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter **endbuf** is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (**endbuf–expbuf**) bytes, a call to **ERROR(50)** is made.

The parameter **eof** is the character which marks the end of the regular expression. For example, in **ed**(1), this character is usually a /.

Each program that includes this file must have a **#define** statement for **INIT**. This definition will be placed right after the declaration for the function **compile** and the opening curly brace ({). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point the beginning of the regular expression so that this register variable can be used in the declarations for **GETC()**, **PEEKC()** and **UNGETC()**. Otherwise it can be used to declare external variables that might be used by **GETC()**, **PEEKC()** and **UNGETC()**. See the example below of the declarations taken from **grep**(1).

There are other functions in this file which perform actual regular expression matching, one of which is the function **step**. The call to **step** is as follows:

```
step(string, expbuf)
```

The first parameter to **step** is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter **expbuf** is the compiled regular expression which was obtained by a call of the function **compile**.

The function **step** returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two

external character pointers are set as a side effect to the call to **step**. The variable set in **step** is **loc1**. This is a pointer to the first character that matched the regular expression. The variable **loc2**, which is set by the function **advance**, points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, **loc1** will point to the first character of **string** and **loc2** will point to the null at the end of **string**.

step uses the external variable **circf** which is set by **compile** if the regular expression begins with ^. If this is set then **step** will try to match the regular expression to the beginning of the string only. If more than one regular expression is to be compiled before the first is executed the value of **circf** should be saved for each compiled expression and **circf** should be set to that saved value before each call to **step**.

The function **advance** is called from **step** with the same arguments as **step**. The purpose of **step** is to step through the **string** argument and call **advance** until **advance** returns non-zero indicating a match or until the end of **string** is reached. If one wants to constrain **string** to the beginning of the line in all cases, **step** need not be called; simply call **advance**.

When **advance** encounters a * or \{ \} sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, **advance** will back up along the string until it finds a match or reaches the point in the string that initially matched the * or \{ \}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer **locs** is equal to the point in the string at sometime during the backing up process, **advance** breaks out of the loop that backs up and returns zero. This is used by **ed**(1) and **sed**(1) for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like **s/y*//g** do not loop forever.

The additional external variables **sed** and **nbra** are used for special purposes.

EXAMPLES

The following is an example of how the regular expression macros and calls look from **grep**(1):

```
#define INIT register char *sp = instring;
#define GETC()      (*sp +)
#define PEEKC()     (*sp)
#define UNGETC(c)   (--sp)
#define RETURN(c)   return;
#define ERROR(c)    regerr()

#include <regexp.h>
...
                (void) compile(*argv, expbuf, &expbuf[ESIZE], '\0');
...
                if (step(linebuf, expbuf))
                            succeed();
```

SEE ALSO

ed(1), expr(1), grep(1), sed(1)

NAME

stat – data returned by stat system call

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
```

DESCRIPTION

The system calls **stat** and **fstat** return data whose structure is defined by this include file. The encoding of the field **st_mode** is defined in this file also.

Structure of the result of stat

```
struct   stat
{
        dev_t     st_dev;
        ino_t     st_ino;
        mode_t    st_mode;
        nlink_t   st_nlink;
        uid_t     st_uid;
        gid_t     st_gid;
        dev_t     st_rdev;
        off_t     st_size;
        timeval_t st_atime;
        timeval_t st_mtime;
        timeval_t st_ctime;
};

#define _ 0170000 /* type of file */
#define _ 0040000 /* directory */
#define _ 0020000 /* character special */
#define _ 0060000 /* block special */
#define _ 0100000 /* regular */
#define _ 0010000 /* fifo */
#define _ 0120000 /* symbolic link */
#define _ 04000   /* set user ID on execution */
#define _ 02000   /* set group ID on execution */
#define _ 01000   /* save swapped text even after use */
#define _ 00400   /* read permission, owner */
#define _ 00200   /* write permission, owner */
#define _ 00100   /* execute/search permission, owner */
#define _ S_ISGID /* record locking enforcement flag */
#define _ 00700   /* read,write, execute: owner */
#define _ 00400   /* read permission: owner */
#define _ 00200   /* write permission: owner */
#define _ 00100   /* execute permission: owner */
#define _ 00070   /* read, write, execute: group */
#define _ 00040   /* read permission: group */
#define _ 00020   /* write permission: group */
```

```
#define _ 00010   /* execute permission: group */
#define _ 00007   /* read, write, execute: other */
#define _ 00004   /* read permission: other */
#define _ 00002   /* write permission: other */
#define _ 00001   /* execute permission: other */
```

SEE ALSO

stat(2), types(5)

NAME

term – conventional names for terminals

DESCRIPTION

These names are used by certain commands (for example **man**(1), **tabs**(1), **tput**(1), **vi**(1) or **curses**(3X)), and are maintained as part of the shell environment in the environment variable **TERM** (see **sh**(1), **profile**(4), and **environ**(5)).

Entries in **terminfo**(4) source files consist of a number of comma-separated fields. (To obtain the source description for a terminal, use the **–I** option of **infocmp**(1M).) White space after each comma is ignored. The first line of each terminal description in the **terminfo**(4) database gives the names by which **terminfo**(4) knows the terminal, separated by bar (|) characters. The first name given is the most common abbreviation for the terminal (this is the one to use to set the environment variable TERMINFO in *$HOME/.profile*; see **profile**(4)), the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks and must be unique in the first 14 characters; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, for example, for the AT&T 4425 terminal, **att4425**. This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Up to 8 characters, chosen from [a–z0–9], make up a basic terminal name. Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name. Terminal sub-models, operational modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. Thus, an AT&T 4425 terminal in 132 column mode would be **att4425–w**. The following suffixes should be used where possible:

Suffix	Meaning	Example
–w	Wide mode (more than 80 columns)	att4425–w
–am	With auto. margins (usually default)	vt100–am
–nam	Without automatic margins	vt100–nam
–n	Number of lines on the screen	aaa–60
–na	No arrow keys (leave them in local)	c100–na
–np	Number of pages of memory	c100–4p
–rv	Reverse video	att4415–rv

To avoid conflicts with the naming conventions used in describing the different modes of a terminal, for example, **–w**, it is recommended that a terminal's root name not contain hyphens. Further, it is good practice to make all terminal names used in the *terminfo*(4) database unique. Terminal entries that are present only for inclusion in other entries via the **use=** facilities should have a '**+**' in their name, as in **4415+nl**

Some of the known terminal names may include the following (for a complete list, type: `ls -C /usr/lib/terminfo/?`):

2621,hp2621 Hewlett-Packard 2621 series

2631	Hewlett-Packard 2631 line printer
2631–c	Hewlett-Packard 2631 line printer - compressed mode
2631–e	Hewlett-Packard 2631 line printer - expanded mode
2640,hp2640	Hewlett-Packard 2640 series
2645,hp2645	Hewlett-Packard 2645 series
3270	IBM Model 3270
33,tty33	AT&T Teletype Model 33 KSR
35,tty35	AT&T Teletype Model 35 KSR
37,tty37	AT&T Teletype Model 37 KSR
4000a	Trendata 4000a
4014,tek4014	TEKTRONIX 4014
40,tty40	AT&T Teletype Dataspeed 40/2
43,tty43	AT&T Teletype Model 43 KSR
4410,5410	AT&T 4410/5410 terminal in 80-column mode - version 2
4410–nfk,5410–nfk	AT&T 4410/5410 without function keys - version 1
4410–nsl,5410–nsl	AT&T 4410/5410 without pln defined
4410–w,5410–w	AT&T 4410/5410 in 132-column mode
4410v1,5410v1	AT&T 4410/5410 terminal in 80-column mode - version 1
4410v1–w,5410v1–w	AT&T 4410/5410 terminal in 132-column mode - version 1
4415,5420	AT&T 4415/5420 in 80-column mode
4415–nl,5420–nl	AT&T 4415/5420 without changing labels
4415–rv,5420–rv	AT&T 4415/5420 80 columns in reverse video
4415–rv–nl,5420–rv–nl	AT&T 4415/5420 reverse video without changing labels
4415–w,5420–w	AT&T 4415/5420 in 132-column mode
4415–w–nl,5420–w–nl	AT&T 4415/5420 in 132-column mode without changing labels
4415–w–rv,5420–w–rv	AT&T 4415/5420 132 columns in reverse video
4415–w–rv–nl,5420–w–rv–nl	AT&T 4415/5420 132 columns reverse video without changing labels
4418,5418	AT&T 5418 in 80-column mode
4418–w,5418–w	AT&T 5418 in 132-column mode
4420	AT&T Teletype Model 4420
4424	AT&T Teletype Model 4424
4424-2	AT&T Teletype Model 4424 in display function group ii
4425,5425	AT&T 4425/5425
4425–fk,5425–fk	AT&T 4425/5425 without function keys
4425–nl,5425–nl	AT&T 4425/5425 without changing labels in 80-column mode
4425–w,5425–w	AT&T 4425/5425 in 132-column mode
4425–w–fk,5425–w–fk	AT&T 4425/5425 without function keys

	in 132-column mode
4425–nl–w,5425–nl–w	AT&T 4425/5425 without changing labels in 132-column mode
4426	AT&T Teletype Model 4426S
450	DASI 450 (same as Diablo 1620)
450–12	DASI 450 in 12-pitch mode
500,att500	AT&T-IS 500 terminal
510,510a	AT&T 510/510a in 80-column mode
513bct,att513	AT&T 513 bct terminal
5320	AT&T 5320 hardcopy terminal
5420_2	AT&T 5420 model 2 in 80-column mode
5420_2–w	AT&T 5420 model 2 in 132-column mode
5620,dmd	AT&T 5620 terminal 88 columns
5620–24,dmd–24	AT&T Teletype Model DMD 5620 in a 24x80 layer
5620–34,dmd–34	AT&T Teletype Model DMD 5620 in a 34x80 layer
610,610bct	AT&T 610 bct terminal in 80-column mode
610–w,610bct–w	AT&T 610 bct terminal in 132-column mode
7300,pc7300,unix_pc	AT&T UNIX PC Model 7300
735,ti	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer
pt505	AT&T Personal Terminal 505 (22 lines)
pt505–24	AT&T Personal Terminal 505 (24-line mode)
sync	generic name for synchronous Teletype Model 4540-compatible terminals

Commands whose behavior depends on the type of terminal should accept arguments of the form **–T term** where **term** is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable **TERM**, which, in turn, should contain **term**.

FILES

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

SEE ALSO

man(1), sh(1), stty(1), tabs(1), tput(1), vi(1), tplot(1G), infocmp(1M), curses(3X), profile(4), terminfo(4), environ(5)

NOTES

Not all programs follow the above naming conventions.

NAME

types – primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

The data types defined in the include file are used in operating system code; some data of these types are accessible to user code:

```
typedef struct _physadr { int r[1]; } *physadr;
typedef struct _quad { long val[2]; } quad;
typedef long        daddr_t;
typedef char *      caddr_t;
typedef unsigned char           unchar;
typedef unsigned short          ushort;
typedef unsigned int            uint;
typedef unsigned long           ulong;
typedef ulong       ino_t;
typedef short       cnt_t;
typedef long        time_t;
typedef long        dev_t;
typedef long        off_t;
typedef long        paddr_t;
typedef int         key_t;
typedef unchar      use_t;
typedef short       sysid_t;
typedef short       index_t;
typedef unchar      lock_t;
typedef uint        size_t;
typedef int         bool_t;
typedef int         spl_t;
typedef long        swblk_t;
typedef short       pid_t;
typedef unsigned short          gid_t;
typedef unsigned short          uid_t;
typedef unsigned short          mode_t;
typedef unsigned char           gate_t;
typedef short       nlink_t;
typedef struct list {
        struct list *l_flink;
        struct list *l_rlink;
        }           list_t;
typedef struct label_t {
        int         lt_esp;
        int         lt_ebp;
        int         lt_eip;
        int         lt_ebx;
        int         lt_esi;
        int         lt_edi;
        } label_t;
typedef  struct {
        int         *s_head;
```

```
        int             *s_tail;
        short           s_count;
        gate_t          s_gate;
        char            s_flags;
        struct  semstat {
                        ulong           ss_visit;
                        ulong           ss_block;
                        ulong           ss_totqlen;
                        }               s_semstat;
        } sema_t;
typedef struct {
        time_t  tv_sec;
        time_t  tv_usec;
        } timeval_t;
```

The form **daddr_t** is used for disk addresses except in an inode on disk, see **fs**(4). Times are encoded in seconds since 00:00:00 UTC, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The **label_t** variables are used to save the processor state while another process is running.

SEE ALSO

fs(4)

NAME

values – machine-dependent values

SYNOPSIS

#include <values.h>

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

BITS(type) The number of bits in a specified type (e.g., int).

HIBITS The value of a short integer with only the high-order bit set (eg 0x8000 for 16 bit shorts).

HIBITL The value of a long integer with only the high-order bit set (eg 0x80000000 for 32 bit integers).

HIBITI The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).

MAXSHORT The maximum value of a signed short integer (eg 0x7FFF ≡ 32767 for 16 bit 2's complement).

MAXLONG The maximum value of a signed long integer (eg 0x7FFFFFFF ≡ 2147483647 for 32 bit 2's complement).

MAXINT The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).

MAXFLOAT, LN_MAXFLOAT
The maximum value of a single-precision floating-point number, and its natural logarithm.

MAXDOUBLE, LN_MAXDOUBLE
The maximum value of a double-precision floating-point number, and its natural logarithm.

MINFLOAT, LN_MINFLOAT
The minimum positive value of a single-precision floating-point number, and its natural logarithm.

MINDOUBLE, LN_MINDOUBLE
The minimum positive value of a double-precision floating-point number, and its natural logarithm.

FSIGNIF The number of significant bits in the mantissa of a single-precision floating-point number.

DSIGNIF The number of significant bits in the mantissa of a double-precision floating-point number.

SEE ALSO

intro(3), math(5)

NAME

varargs – handle variable argument list

SYNOPSIS

#include <varargs.h>

va_alist

va_dcl

void va_start(pvar)
va_list pvar;

type va_arg(pvar, type)
va_list pvar;

void va_end(pvar)
va_list pvar;

DESCRIPTION

This set of macros allows portable procedures that accept variable argument lists to be written. Routines that have variable argument lists (such as **printf**(3S)) but do not use **varargs** are inherently nonportable, as different machines use different argument-passing conventions.

va_alist is used as the parameter list in a function header.

va_dcl is a declaration for **va_alist**. No semicolon should follow **va_dcl**.

va_list is a type defined for the variable used to traverse the list.

va_start is called to initialize **pvar** to the beginning of the list.

va_arg returns the next argument in the list pointed to by **pvar**. **type** is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, as it cannot be determined at runtime.

va_end is used to clean up.

Multiple traversals, each bracketed by **va_start** ... **va_end**, are possible.

EXAMPLES

This example is a possible implementation of **execl**(2).

```
#include <varargs.h>
#define MAXARGS 100

/* execl is called by
    execl(file, arg1, arg2, ..., (char *)0);
*/
execl(va_alist)
va_dcl
{
  va_list ap;
  char *file;
  char *args[MAXARGS];
  int argno = 0;

  va_start(ap);
```

```
    file = va_arg(ap, char *);
    while ((args[argno++] = va_arg(ap, char *)) != (char *)0)
      ;
    va_end(ap);
    return execv(file, args);
}
```

SEE ALSO

exec(2), printf(3S), vprintf(3S)

NOTES

It is up to the calling routine to specify how many arguments there are, since it is not always possible to determine this from the stack frame. For example, **execl** is passed a zero pointer to signal the end of the list. **printf** can tell how many arguments are there by the format.

It is non-portable to specify a second argument of **char**, **short**, or **float** to **va_arg**, since arguments seen by the called function are not **char**, **short**, or C converts **char** and **short** arguments to **int** and converts **float** arguments to **double** before passing them to a function.

NAME

intro – introduction to special files

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals, and operating system device drivers. STREAMS (see **intro**(2)) software drivers, modules and the STREAMS-generic set of **ioctl**(2) system calls are also described.

For hardware related files, the names of the entries are generally derived from names for the hardware, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding operating system device driver are discussed where applicable.

Disk device filenames are in the following format:

/dev/[r]dsk/<*driver-name*><*drive-number*>[**s**<*number*>]

where **r** indicates a raw interface to the disk, and **s**<*number*> indicates the section (slice) number of the partitioned device. For example, the root partition on SCSI device 0 might be:

/dev/dsk/sd0s0

NAME

clone – open any minor device on a STREAMS driver

CONFIGURATION ENTRY

pseudodevice clone

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 driver indicated is not a STREAMS driver.

WARNINGS

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)

NAME

cmpt – compatibility partitioning driver

CONFIGURATION ENTRY

pseudodevice cmpt

DESCRIPTION

This device driver provides compatibility disk partitioning, meaning that access is provided to the underlying disk space through partitions statically defined in the kernel. The partitions cannot be changed unless the kernel is relinked. The partition layout is compatible with that used by DYNIX V3.0. This is a *virtual* device driver, in that it requires an actual disk device (such as a **zd**, **sd**, or **wd** device) to build upon.

Disk device drivers do not provide any partitions, but present the entire disk surface for use. To make this so-called *unpartitioned* disk more useful, the disk surface may be partitioned by the **cmpt** driver into up to 8 possibly overlapping sections. Each of these disk sections is accessed with a separate minor number and each of these sections matches the disk partitions which were the default in DYNIX V3.0.

Special files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 256 through 263 refer to drive 1, and so forth. The special files with minor number 255 refers to drive 0 in the unpartitioned state, and is not accessible if a partitioning driver is in place on this disk. (The jump in minor numbers between drive 0 and drive 1 is due to support for the **vtoc** driver, which can support up to 255 partitions per disk.) The standard device names begin with the disk controller name (e.g., **zd** or **wd**), followed by the drive number, followed by the letter **s**, and then a number from 0-7 — e.g., **zd0s0**. By convention, these special files are found in the directory */dev/dsk*.

The block devices access the disk through the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a *raw* interface that provides for direct transmission between the disk and the user's read or write buffer. Raw transfers are limited to a maximum raw transfer size. The maximum raw transfer size is specific to each driver, but is usually 256 512-byte sectors. In raw I/O, counts must be a multiple of 512 bytes, and must begin on 16-byte aligned memory addresses. Likewise, **lseek** calls should specify a multiple of 512 bytes. A single read or write call usually results in exactly one I/O operation, which transfers data directly to the user's address space. Therefore, raw I/O is considerably more efficient when many words are transmitted. The names of the raw special files are conventionally the same as the block special files, and are found in the directory **/dev/rdsk** — e.g., */dev/rdsk/wd0s0*. In addition to these regular device interfaces, there is a *diagnostic* interface to the disk once the **cmpt** driver is present. This interface allows **cmpt** ioctls to be issued and also provides a read-only interface to the entire unpartitioned disk address space. The special file for the diagnostic interface has a major and minor device number equal to the unpartitioned device with 32768 added to the major number. By convention, this special file is in the directory */dev/diag/rdsk*— for example, */dev/diag/rdsk/sd0*.

Note that when the **cmpt** driver is in place on a disk, the unpartitioned interface to the disk becomes *unavailable*, meaning that all attempts to open the unpartitioned disk fail with ENXIO.

BINARY CONFIGURATION

When building a kernel with the **cmpt** driver present, be sure that the binary configuration information in */etc/conf/uts/io/cmpt/cmpt_space.c* is up to date. This includes partition tables for all of the disks which were supported under DYNIX V3.0. Other drives may be added to the configuration tables. The origin in sectors and size in sectors of the partitions on each drive are as follows:

Vertex model 170

partition	start	length
0	0	15884
1	15884	33440
2	0	117096
3	49324	15884
4	65208	51888
5	0	0
6	49324	67772
7	0	0

Fujitsu model 2243AS

partition	start	length
0	0	15884
1	15884	33440
2	0	140436
3	49324	15884
4	65208	75228
5	0	0
6	49324	91112
7	0	0

Maxtor model XT1140

partition	start	length
0	0	15884
1	15884	33440
2	0	233324
3	49324	15884
4	65208	168116
5	0	0
6	49324	184000
7	0	0

CDC Wren III 94161

partition	start	length
0	0	15884
1	15884	33440
2	0	303975
3	49324	15884
4	65208	238767
5	0	0
6	49324	254651
7	0	0

Micropolis 1375

partition	start	length
0	0	15884
1	15884	33440
2	0	284480
3	49324	15884
4	65208	219272
5	0	0
6	49324	235156
7	0	0

Fujitsu 2246sa

partition	start	length
0	0	15884
1	15884	33440
2	0	270270
3	49324	15884
4	65208	205062
5	0	0
6	49324	220946
7	0	0

Fujitsu M2333K (swallow)

partition	start	length
0	221100	16500
1	237600	67320
2	660	541200
3	660	270600
4	271260	270600
5	660	236940
6	304920	236940
7	660	220440

Fujitsu M2351A (Eagle)

partition	start	length
0	336720	16560
1	353280	67160
2	920	771880
3	920	385480
4	386400	385480
5	920	352360
6	420440	352360
7	920	335800

Fujitsu M2344K (swallow 4)

partition	start	length
0	502524	17820
1	520344	67716
2	1782	1106622
3	1782	552420
4	554202	554202
5	1782	518562
6	588060	520344
7	1782	500742

Fujitsu M2382K (swallow 5)

partition	start	length
0	761076	17496
1	778572	69984
2	2187	1622754
3	2187	811377
4	813564	811377
5	2187	776385
6	848556	776385
7	2187	758889

CDC 9715-340 (FSD)

partition	start	length
0	238272	17136
1	255408	67728
2	816	577728
3	816	288864
4	289680	288864
5	816	254592
6	323136	255408
7	816	237456

CDC 9771-800 (XMD)

partition	start	length
0	644640	16320
1	660960	68000
2	1360	1388560
3	1360	694960
4	696320	693600
5	1360	659600
6	728960	660960
7	1360	643280

It is unwise to use two overlapping partitions simultaneously, since this can cause corruption of data.

IOCTLS

This driver supports the following commands to **ioctl()**:

V_BUILD

This ioctl is used to build a cmpt device from an unpartitioned device. It should be passed to the device normally called **/dev/cmpt** and have, as the third argument to the system call, a pointer to a filled-out *struct cmptdt*. This structure is defined in */usr/include/sys/cmpt.h*. The fields of *struct cmptdt* are *cd_fd*, an open file descriptor for a raw disk that does not have a partitioning driver in place; and *cd_type*, an integer index into the table *cmptparts* defined in */etc/conf/uts/io/cmpt/cmpt_space.c*. A possible coding example using this ioctl would be:

```
struct cmptdt cd;

fd = open("/dev/cmpt", O_RDWR);
fd1 = open("/dev/rdsk/zd1", O_RDWR);
cd.cd_fd = fd1;
cd.cd_type = 6;  /* M2333K */
ioctl(fd, V_BUILD, &cd);
```

This shows partitioning being added to disk type 6 (a Fujitsu "Swallow" 3, model "M2333K"). If the ioctl is successful, the file descriptor in *cd_fd* should be treated as if it has already been closed and a subsequent **close()** operation is not needed on this descriptor. This is necessary since a successful V_BUILD operation will make the unpartitioned device inaccessible (i.e., */dev/rdsk/zd1* can no longer be opened if the V_BUILD operation above is successful).

V_DESTROY

The cmpt device in place on a disk is instructed to destroy all of the devices associated with partitions on this disk. The unpartitioned disk device is made available for use again. This ioctl will fail with EBUSY if any of the partitions which make up this disk are open, being used as a swap device, or has a mounted filesystem. The semantics of this ioctl are identical to the V_BUILD ioctl, with the exception that the third argument to the ioctl call is the file descriptor returned from opening any valid partition on this disk or from opening the diagnostic device for this disk.

V_WRITEPROTECT

This ioctl acts as a software write-protect switch. The first argument to the ioctl call should be an open file-descriptor to a raw partition. When V_WRITEPROTECT is run, all subsequent attempts to open for writing the character-special or block-special device for this partition will result in EBUSY being returned in errno. Note that the *mount*(2) system call performs the equivalent of a V_WRITEPROTECT ioctl to the mounted device.

V_WRITEENABLE

This ioctl reverses the effect of the V_WRITEPROTECT ioctl, allowing the partition's character and block-special devices to be opened for writing.

DEVTAB ENTRY

In order for the **cmpt** driver to get established on disks automatically at system boot time, the file */dev/devtab* must have lines added which direct the binding of the **cmpt** driver to disks. Lines for **cmpt** in */etc/devtab* have the following form:

cmpt special disktype

where *special* is a character special file for an unpartitioned physical disk device, and *disktype* describes the type of disk. The valid disk type names are listed in the file */etc/devbuild.d/cmpt.types*. If the **cmpt** driver becomes successfully established on the disk, *special* will become unavailable for use and

the partitioned devices themselves become useable.

For example, consider the following lines from */etc/devtab*:

```
cmpt   /dev/rdsk/zd1      m2333k
cmpt   /dev/rdsk/zd2      m2344k
cmpt   /dev/rdsk/sd0      wren3
```

These lines will automatically build a cmpt driver on **zd1**, **zd2**, and **sd0**. The partitions for **zd1** will be those for an m2333k (a Fujitsu "Swallow 3"); partitions for **zd2** will be those for an m2344k (a Fujitsu "Swallow 4"); and partitions for **sd0** will be those for a CDC Wren 3.

FILES

/dev/cmpt
/dev/dsk/*
/dev/rdsk/*
/dev/diag/rdsk/*
/etc/devtab
/etc/devbuild.d/cmpt.types

SEE ALSO

vtoc(7), devbuild(1a), devdestroy(1a), devtab(4)

NAME

cn – pseudoconsole driver

CONFIGURATION ENTRY

pseudodevice cn

DESCRIPTION

The purpose of the **cn** driver is to give users of the */dev/console* device (e.g., daemons) a device which is immune to job control interference, and which tracks the selected console device on either SSM or SCED.

Special Files

This driver creates three special files:

/dev/syscon tracks the console device which is selected at system boot time. It is used for single-user shells, and logins. Job control on this device does not affect processes that are writing to */dev/console*. */dev/syscon* supports job control.

/dev/altcon is the "other" port on the SCED or SSM console port. Is useful for interactive purposes such as logins. */dev/altcon* supports job control.

/dev/console is meant to be used only by daemons and for logging text to the console device which was selected at system boot time. It tracks the console device, but does not allow processes opening it to become controlling processes. This way, these daemon processes don't interfere with users of /dev/syscon who *want* to become controlling processes. */dev/console* does not support job control.

Usage

Writes to */dev/console* pass an extra flag to the stream code which forces it to *not* treat processes writing to */dev/console* as background processes. Without this treatment, these processes could be *stopped* if they tried to write. Reads and IOCTLs to this special file are disallowed. They return with an EBADF error code.

DIAGNOSTICS

cn_init: invalid device init. An invalid device initialization was attempted in the cn driver. This indicates a kernel software bug.

cn_init: real cn dev hasn't been added. The cn driver initialization code failed because the real console driver hasn't been established yet. This could be either a kernel software error or kernel configuration error.

cn: problem adding syscons dev to devsw0 A problem occurred in trying to add the syscon tty device to the system. This is probably due to a driver which has added a device with the same major/minor number.

SEE ALSO

console(7)

NAME

consmem, sm, ss - console memory drivers

CONFIGURATION ENTRY

pseudodevice consmem

DESCRIPTION

consmem is a special file that is a pseudo driver that selects either the **sm** or **ss** console memory driver, depending on the location of the console. This allows system-independent code to be developed. Normally, all references to the console memory device should go through **consmem.**

sm is a special file that is an image of the memory located on the SCED board, and **ss** is a special file that is an image of the memory located on the SSM board. They may be used, for example, to get system messages that are stored in battery-backed-up RAM.

The layout of the address space depends on which console memory driver is actually in use by the system.

Examining and patching device registers is likely to lead to unexpected results in both the **sm** and **ss.**

Consmem is a special device that always accesses the SCED or SSM board that is attached to the front panel and console.

The following *ioctl* functions are provided:

SMIOSTATS	Get statistics about the use of the **consmem, sm** or **ss** driver
SMIOGETREBOOT0	Get the standard reboot structure
SMIOSETREBOOT0	Set the standard reboot structure
SMIOGETREBOOT1	Get the dump reboot structure
SMIOSETREBOOT1	Set the dump reboot structure
SMIOGETLOG	Get data regarding the system log
SMIOSETLOG	Set data regarding the system log (SCED only)

SMIOSTATS uses the structure:

```
struct consmem_stats {
   int  consmem_cmd;     /* number of commands */
   int  consmem_ioctls;  /* number of ioctls */
   int  consmem_xfers;   /* number of data transfers */
};
```

SMIOGETREBOOT0, SMIOSETREBOOT0, SMIOGETREBOOT1 and SMIOSETREBOOT1 use the structure:

```
struct ioctl_reboot {
   unsigned char   re_powerup;   /* reference powerup defaults */
   unsigned int    re_boot_flag; /* Boot flags */
   unsigned char   *re_cfg_addr; /* address to build config */
   char      re_boot_name[BNAMESIZE];    /* Boot image name */
};
```

There are two boot names contained in the firmware. They are *boot_name*[0] and *boot_name*[1]. The SMIOGETREBOOT0 and SMIOSETREBOOT0 get and set *boot_name*[0]. The SMIOGETREBOOT1 and SMIOSETREBOOT1 get and set *boot_name*[1]. The value of the **re_powerup** flag must always be set to 0 before calling the kernel. For the SCED, there are also two **re_cfg_addr** values. SMIOGETREBOOT0 and SMIOSETREBOOT0 access the first value and SMIOGETREBOOT1 and SMIOSETREBOOT1 access the second. The fields, *boot_name*[0] and *boot_name*[1] are stored in battery-backed-up RAM and exist across power cycles. If they are changed and the system rebooted the new names are used.

SMIOGETLOG and SMIOSETLOG (SCED only) use the structure:

```
struct cons_mem {
   char *mm_buffer;     /* address of log buffer */
   char *mm_nextchar;   /* next free char in buffer */
   short mm_size;       /* buffer size */
   short mm_nchar;      /* number valid chars in buf */
};
```

A program can use this structure to determine the location and size of the log buffer and then use the read system call to get the actual data.

FILES

/dev/consmem
/dev/cmem/sm*
/dev/cmem/ss*

SEE ALSO

dmesg(1M), savecore(1M), bootflags(1M)

BUGS

The **sm** driver reads values as bytes; this may not be appropriate for some 16-bit device registers on the board.

NAME

console – console interface

DESCRIPTION

The Sequent console interface provides access to the system console through a Sequent SCSI/Ethernet/Diagnostics (SCED) Board or Sequent System Services Module (SSM), depending on which board has taken control of the system during boot. (If an SSM is present in a system, it will provide the console interface, even though a SCED board may also be present in the system. If no SSM is present, the SCED board will provide the console interface.)

The **co** driver provides the SCED console interface functionality. Similarly, the **sc** driver provides the SSM console interface functionality.

When the operating system is running, the console interface behaves like a normal terminal; see **streamtty**(7). Two terminal lines are available per SCED or SSM. All standard baud rates are supported on each of these lines. The hardware signals DTR and RTS are enabled on open of the lines and disabled on close if HUPCL is specified. These signals may be changed via ioctl calls to the device driver and are used when flow control is enabled. Break character detection and generation is also supported.

Selection of the system console and the alternate console are determined at boot time by the monitor firmware based on the LOCAL and REMOTE front panel switch. The system console is the selected port and the alternate console is the unselected port. Device special files named */dev/console* and */dev/syscon* will track the system console port. The special file named */dev/altcon* represents the alternate console port. */dev/console* is a **write-only** device that can be used to log output from background daemons to the system console device with no consequence of becoming the controlling tty. */etc/getty* processes should be put on */dev/syscon* and */dev/altcon* for system login purposes but will fail when put on */dev/console*.

If the SCED monitor firmware flags specify that break characters cause a return to monitor firmware, a break on a SCED console line will cause an immediate return to firmware. On all other SCED console lines and all SSM console lines, the break character is interpreted according to **streamtty(7).**

Other than these exceptions, the system console and the alternate console work like any other terminal line.

Minor Numbers

Minor numbers for device nodes corresponding to console lines are selected as follows:

SCED Device	Minor Number	Location
/dev/ttyBA/BAAA	0	SCED Board 0, local port
/dev/ttyBA/BAAB	1	SCED Board 0, remote port
/dev/ttyBA/BAAC	2	SCED Board 1, local port
/dev/ttyBA/BAAD	3	SCED Board 1, remote port

/dev/ttyBA/BAAE	4	SCED Board 2, local port
/dev/ttyBA/BAAF	5	SCED Board 2, remote port
/dev/ttyBA/BAAG	6	SCED Board 3, local port
/dev/ttyBA/BAAH	7	SCED Board 3, remote port

SSM Device	Minor Number	Location
/dev/ttyWA/WAAA	0	SSM Board 0, local port
/dev/ttyWA/WAAB	1	SSM Board 0, remote port
/dev/ttyWA/WAAC	2	SSM Board 1, local port
/dev/ttyWA/WAAD	3	SSM Board 1, remote port
/dev/ttyWA/WAAE	4	SSM Board 2, local port
/dev/ttyWA/WAAF	5	SSM Board 2, remote port
/dev/ttyWA/WAAG	6	SSM Board 3, local port
/dev/ttyWA/WAAH	7	SSM Board 3, remote port

FILES

/dev/console
/dev/syscon
/dev/altcon
/dev/ttyBA/BAA[A-H]
/dev/ttyWA/WAA[A-H]
/etc/conf/uts/io/co/co_space.c
/etc/conf/uts/io/sc/sc_space.c

SEE ALSO

streamtty(7), cn(7)

NAME

kMBmem, kMWmem – MULTIBUS memory

CONFIGURATION ENTRY

pseudodevice kmmem

DESCRIPTION

The files */dev/cmem/kMBmem* and */dev/cmem/kMWmem* represent MULTIBUS address space, consecutively mapped 1 megabyte per MULTIBUS controller, starting at 0 for the first controller. (Deconfigured controllers still reserve their own address space.) Accesses via */dev/cmem/kMBmem* are one byte at a time, while accesses via */dev/cmem/kMWmem* are two bytes at a time (one 16-bit word).

A mapping request with **mmap**(2seq) cannot cross an MBAd boundary, and must be made to an existing MBAd. There is no checking to prevent a mapping request from referencing non-existent MBAd address space. These devices map as physical maps (MM_PHYS).

FILES

/dev/cmem/kMBmem
/dev/cmem/kMWmem

SEE ALSO

mmap(2seq)

NAME

ld0 – STREAMS tty line discipline

DESCRIPTION

ld0 is a STREAMS module that implements most of the terminal interface described in **termio**(7). It relies on the downstream driver supporting the generic STREAMS tty driver interface described in **streamtty**(7).

Read-side Received Messages

M_CTL The first data byte of the control message indicates the type of operation. Additional data bytes may be supplied dependent on the operation. Supported operations are:

MC_FRAME_ERR A framing error was detected. If the data value of the character with the framing error is known, it is provided in the remaining data bytes of the message. If not supplied, the data value of the character is assumed to be zero. A framing error character whose data value is zero is interpreted as a break condition. If CREAD (*c_cflag*) is not set, the message is discarded. Otherwise the message is entirely processed within this module. In either case the message is not passed upstream.

MC_PARITY_ERR

A parity error was detected. If the data value of the character with the parity error is known, it is provided in the remaining data bytes of the message. If not supplied, the data value of the character is assumed to be zero. If CREAD (*c_cflag*) is not set, the message is discarded. Otherwise the message is entirely processed within this module. In either case the message is not passed upstream. If INPCK (*c_iflag*) is set, the data in the message is processed as a character with a parity error; otherwise the message is treated as an M_DATA message (that is, as a character without a parity error).

M_PCCTL The first data byte of the control message indicates the type of operation. Additional data bytes may be supplied dependent on the operation. Supported operations are:

MC_STOPI The driver is requesting that all input data messages stop if IXOFF is set. Typically a **MC_STOPI** message is sent back to the driver.

M_DATA If CREAD (*c_cflag*) is set, several kinds of input processing are done according to **termio**(7); these are briefly described below. If CREAD is not set, the message is discarded.

If ISIG is set, INTR, QUIT, and SUSP characters are immediately detected and result in an M_PCSIG message sent upstream corresponding to the appropriate signal.

If IXON is set, M_PCCTL messages requesting the MC_START and MC_STOP functions (see TCXONC below) are sent downstream in immediate response to the START and STOP characters, respectively. If IXANY is also set, any character can function as a START character.

In canonical mode (ICANNON is set), received data is assembled into lines with erase, kill, and other processing as described in **termio**(7). The buffer used for line assembly can contain at least a maximum of MAX_CANON characters; if it overflows, all characters in it are silently thrown away.

In non-canonical mode, received data is not internally retained or delayed for line assembly but is rather immediately passed upstream. The semantics of MIN and TIME (as described in **termio**(7)) are implemented at the stream head.

M_FLUSH If FLUSHR or FLUSHRW is specified, the read queue is flushed along with all the data collected in the current line and the message is passed upstream.

M_IOCACK The message data is augmented (in the case of TCGETA and TCGETP) and the message is passed upstream. If the *ioc_id* matches that saved on an incoming TIOCGETP, it is converted to a struct **sgttyb** before being sent upstream as a TIOCGETP.

M_IOCNAK If the message is for a TIOCGWINSZ or TIOCSWINSZ **ioctl**(2) call, the M_IOCNAK is intercepted and the **ioctl** is performed as described in the IOCTLS sections below.

All other messages are passed upstream unchanged.

Write-side Received Messages

M_DATA If OPOST is set, or if ICANON, XCASE, and IEXTEN are set, appropriate output processing is performed on the data and the result is passed downstream, possibly augmented with M_DELAY messages. Otherwise, the data is passed downstream unchanged.

M_FLUSH If FLUSHW or FLUSHRW is specified, the write queue is flushed and the message is passed downstream.

M_IOCTL
M_PCIOCTL The **ioctl**(2) calls described in **termio**(7) that are supported by **ld0** are described in the IOCTLS section below. Note that the TIOCGWINSZ and TIOCSWINSZ **ioctl** calls are not recognized at this point, but later if an M_IOCNAK message is received on the read-side.

M_IOCDATA Recognized if it applies to a TRANSPARENT **ioctl** being currently performed.

All other messages are passed downstream unchanged.

Additional Read-side Processing

When the **ld0 put** procedure determines that flow control is trying to prevent additional messages to **ld0** (via **canput**()), the message is enqueued and, if IXOFF (*c_iflag*) is set, a request is sent to the device driver to suspend additional input. This request is an M_PCCTL message sent downstream whose first data byte is MC_STOPI and remaining data bytes contain the current STOP character. When the **service** procedure determines that flow control is allowing additional messages to **ld0** and IXOFF processing currently has input suspended, a request is sent to the device driver to resume suspended input. This request is an M_PCCTL message sent downstream whose first data byte is MC_STARTI and remaining data bytes contain the current START character.

IOCTLS

TCGETA
TCGETP
TIOCGETP M_PCIOCTL message passed downstream during write-side processing. If a positive acknowledgment is seen, the parameter information provided by the driver and downstream modules is augmented and passed upstream.

In the case of **TIOCGETP** the incoming struct **sgttyb** is converted into a **termios** structure and the ioctl message type is changed to **TCGETP** and the *io_id* is saved. When the M_IOCACK is returned, it is converted back to its original form.

TIOCGWINSZ M_IOCNAK message received during read-side processing. The negative acknowledgement is intercepted and the **ioctl** operation is performed. If the message is for a TRANSPARENT **ioctl**, an M_COPYOUT message is sent upstream containing a struct **winsize** with the current setting of the window size of the terminal. If the message is not a TRANSPARENT **ioctl**, the struct **winsize** is sent up in an M_IOCACK message.

TCSETP
TCSETPW
TCSETPF
TCSETA
TCSETAW
TCSETAF
TIOCSETP Recognized during write-side processing. The function is partially performed by **ld0** and the message is passed downstream. In the case of TCSETP and TCSETA the message is an M_PCIOCTL otherwise the message is an M_IOCTL.

The parameters relevant to **ld0** operation are internally updated and the message is passed downstream unmodified. If a parameter change requires options at the stream head to change, an M_PCSETOPT message is sent upstream. If ICANON is set, the stream head is placed in message non-discard mode and the MIN and TIME stream head options are set to 1 and zero, respectively. If ICANON is not set, the stream head is placed in byte-stream mode and the MIN and TIME stream head options are set from the values in the **ioctl**. In either case, if TOSTOP (*c_lflag*) is changed in the **ioctl**, the stream head option is set based on the value in the **ioctl**.

All *c_cflag* fields except CREAD are not implemented by **ld0** and hence must be supported by lower-level modules in order to fully implement the **termio**(7) interface.

Other parameters, while implemented by **ld0**, may also be interpreted by lower-level modules in an effort to provide more intelligent or timely behavior as long as this behavior adheres to the **termio**(7) interface. For example: If IXON (*c_iflag*) is set, lower modules could do output flow control (e.g., in hardware) using the START and STOP characters defined in the *c_cc* array rather than relying on **ld0** to send M_PCCTL messages requesting the MC_START and MC_STOP functions downstream (see TCXONC below) in response to the START and STOP characters; IXANY (*c_iflag*) could be similarly handled. If CREAD (*c_cflag*) is not set, the terminal receiver hardware could be disabled rather than relying on **ld0** to silently discard input in this case.

In the case of TIOCSETP, the incoming struct **sgttyb** is converted into a **termios** structure and its type changed to **TCSETP**. When the M_IOCACK is returned it is converted back to its original form.

TCXONC M_PCIOCTL message recognized during write-side processing. The function is entirely performed by **ld0**; the reply is sent upstream and no message is passed downstream.

If the argument is zero, an M_PCCTL message is sent downstream whose first data byte is MC_STOP. If the argument is 1, an M_PCCTL message is sent downstream whose first data byte is MC_START. If the argument is 2, an M_PCCTL message is sent downstream whose first data byte is MC_STOPI and remaining data bytes contain the current STOP character. If the argument is 3, an M_PCCTL message is sent downstream whose first data byte is MC_STARTI and remaining data bytes contain the current START character.

TCFLSH M_PCIOCTL message recognized during write-side processing. The function is entirely performed by **ld0**; the reply is sent upstream and no message is passed downstream.

If the argument is zero, an M_FLUSH message is sent downstream whose first data byte is FLUSHR. If the argument is 1,

an M_FLUSH message is sent upstream whose first data byte is FLUSHW. If the argument is 2, an M_FLUSH message is sent downstream whose first data byte is FLUSHR and an M_FLUSH message is sent upstream whose first data byte is

Whenever read-side flushing is requested, the M_IOCACK is delayed until after the read side has processed the M_FLUSH such that the flush is synchronous with respect to subsequent (but not concurrent) data reads.

TIOCSWINSZ M_IOCNAK message received during read-side processing. The negative acknowledgement is intercepted and the TRANSPARENT **ioctl** operation is performed. An M_COPYIN message is sent upstream requesting the struct **winsize** with the new settings for the window size of the terminal. When the subsequent M_IOCDATA message is received, the terminal window size parameters are adjusted accordingly. If any of the current settings are changed, an M_SIG message is sent upstream causing the **SIGWINCH** signal to be generated.

WARNINGS

The following **termio**(7) functionality is not implemented by **ld0** or the terminal driver and needs to be implemented at or above the stream head.

Support in **read**(2), **write**(2), **ioctl**(2), and **fcntl**(2) for: Terminal Access Control functions associated with job control (foreground process group checks including TOSTOP), MIN and TIME semantics in non-canonical mode, and O_NONBLOCK and O_NDELAY semantics. The TIOCGPGRP and TIOCSPGRP **ioctls** in their entirety. Modem Disconnect semantics triggered on receipt of an M_HANGUP message.

FILES

/dev/tty*

SEE ALSO

stty(1), fork(2), ioctl(2), setpgrp(2), signal(2), streamtty(7), termio(7)

NAME

log – interface to STREAMS error logging and event tracing

CONFIGURATION ENTRY

pseudodevice log *number*

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. The driver supports *number* minor devices, which limits the number of user processes that can simultaneously be submitting **log** messages. (Note that two of those devices will be used by the error logger and trace logger if those processes are running.)

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 The STREAMS module ID number for the module or driver submitting the *log* message.

sid An internal sub-ID number usually used to identify a particular minor device of a driver.

level A tracing level that allows for selective screening out of low-priority messages from the tracer.

flags 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 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 recieve **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";

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

clone(7), getmsg(2), intro(2), putmsg(2), strace(1M), strerr(1M)

NAME

lp – Systech parallel line printer interface

CONFIGURATION ENTRY

lp on multibus *multibus* **unit** *unit* **segsize 1024 nsegs 4**

DESCRIPTION

The **lp** device driver supports the Systech MLP-2000 controller for line printers using the Centronics or Dataproducts parallel interfaces. It supports up to two line printers per controller. The standard controller names consist of **lp** followed by the controller unit number (e.g., **lp0**, **lp1**), which specifies the hardware configuration of the printer.

The controller board, on MULTIBUS *multibus*, is configuration unit number *unit*. The configuration unit number is an index into the *lp_multibus_config* table in the file */etc/conf/uts/io/lp/lp_space.c* and must be unique with respect to other lp devices on this MULTIBUS. Both the *multibus* and *unit* may be specified with a "?," meaning they are wild-carded. If *multibus* is wild-carded, the system assigns the number of the MULTIBUS in which the board is physically located. If *unit* is wild-carded, the system assigns the configuration unit number of the first unused matching entry in the *lp_multibus_config* table. A board matches if it is at the address specified in the *lp_multibus_config* table entry. The interrupt vector cannot be checked, so care must be taken to ensure that the interrupt vector is jumpered to match the *lp_multibus_config* table entry. The *segsize* field is the size of address space reserved for DMA transfers and must be 1024. The *nsegs* field indicates how many regions to reserve for DMA transfers for this device and must be 4.

The minor device number addresses a specific printer by controller and port number. Minor numbers 0 and 1 indicate ports 0 and 1, respectively, on the first controller in the system configuration file, minor numbers 2 and 3 indicate ports 0 and 1, respectively, on the second controller in the system configuration file, and so forth. The standard device names consist of **lp** followed by the minor device number (e.g., **lp0**, **lp1**). Each device is exclusive access on open.

Each controller occupies 32 contiguous bytes of MULTIBUS I/O address space starting at the **csr** address.

BINARY CONFIGURATION

The printer can be configured for one of three different modes which are described in **lpmod**(7). The different modes are implemented by **lpmod**, a STREAMS modules pushed onto the driver automatically when the device is opened. If the system is configured to not have this module pushed, the printer will always be in *raw* mode regardless of what is in the *lp_config* table. The table controlling the push for this module is in */etc/conf/uts/kernel/i386_space/autop_space.c*.

Raw mode provides an eight-bit data path to the printer. Tabs, formfeeds, newlines and other special characters are not expanded, and no line or page adjustments are made.

To see a description of the other modes, refer to the man page for **lpmod**.

The configuration options are specified in a table in the binary configuration file, */etc/conf/uts/io/lp/lp_space.c*. The first column specifies the line

length. The second column specifies the number of lines per page. The third column specifies the mode. Note that all of these fields are ignored by the driver and are only there for the benefit of modules which may be pushed onto the driver. The printer characteristics are indexed by the minor number of the device. If the minor number is greater than the number of entries in the array, the configuration options for the last entry in the table are used. For example, if you had the following hardware configuration:

```
printer 0    80 columns  25 lines upper case only
printer 1   132 columns  25 lines raw
printer 2   132 columns  96 lines raw
printer 3    80 columns  66 lines raw
printer 4   132 columns  66 lines special processing
printer 5   132 columns  66 lines special processing
```

you could use the following configuration table:

```
struct lp_printer  lpconfig[] = {

/*    cols, ht,   special_map
                                    */
{     80,   25,   PRCAPS       } ,   /* 0 */
{     132,  25,   PRRAW        } ,   /* 1 */
{     132,  96,   PRRAW        } ,   /* 2 */
{     80,   66,   PRRAW        } ,   /* 3 */
{     132,  66,   PRSPECIAL    } ,   /* 4 */
};
```

Note that the sixth printer (number 5) is not specified in the configuration but will have the same characteristics as printer number 4.

The default configuration is as follows:

```
struct lp_printer  lpconfig[] = {

/*    cols, ht,   special_map
                                */
{     132,  66,   PRRAW } ,         /* 0 */
{     132,  66,   PRRAW } ,         /* 1 */
};
```

Buffer Size

The number of maps specified in the entry for the controller in */etc/conf/io/lp_space.c* determines the size of the data buffer. The maximum buffer size per printer is 8K bytes. Each printer is allocated half the number of maps specified for its controller (since each controller supports up to two printers). Each map results in 1K bytes of buffering.

FILES

/dev/printer/lp*
/etc/conf/uts/io/lp/lp_space.c

SEE ALSO

lp(1), lpmod(7), lpstat(1), pr(1), lpadmin(1M), terminfo(4)

NAME

lpmod – parallel printer canonical processing module

DESCRIPTION

lpmod is a STREAMS module that provides special character processing for parallel printers. It supports three modes of processing, *raw* mode, *special processing* mode, and *uppercase-only* mode. **lpmod** requests configuration information from the driver in order to determine which mode to use. It expects to get a line length value as well as the mode from the driver. If the driver does not support the request message, *lpmod* uses default values: 132 for the line length and PRRAW (raw mode) for the mode.

PRRAW mode

Raw mode provides an eight-bit data path to the printer. Tabs, formfeeds, newlines and other special characters are not expanded, and no line or page adjustments are made. Raw mode is essentially the same as not having **lpmod** pushed on the stream; it passes data to the driver unchanged.

PRSPECIAL mode

In special processing mode, **lpmod** interprets newlines, backspaces, tabs, and form feeds. Tab stops are every 8 columns starting at column 8 (i.e., 8, 16, 24...). Most printers can operate in both print and plot modes using the special processing mode. In addition, **lpmod** outputs a formfeed on close, and truncates lines longer than the length specified in the printer configuration table. Lines longer than the maximum line length are silently truncated.

PRCAPS mode

In uppercase-only mode, **lpmod** performs all the functions specified above for special processing mode, and in addition does some character mapping. In particular, lower case letters are mapped into upper case. Other characters are mapped to similar characters overstruck with "-" according to the following table:

Input	Mapped to
{	(
}	)
`	'
|	+
~	^

CONFIGURATION

The system can be configured to push **lpmod** automatically onto a driver when a device is opened. The table enabling this is in */etc/conf/uts/kernel/i386_space/autop_space.c*.

Alternatively, a user process opening a device can push **lpmod** using the STREAMS push mechanism.

See individual driver man pages to determine how to set the mode and line length values for a specific device.

FILES

/dev/printer/*
/etc/conf/uts/kernel/i386_space/autop_space.c

SEE ALSO

lp(7), terminfo(4)

NAME

mbad – MULTIBUS adapter

CONFIGURATION ENTRY

mbad *interface* **on sb8000 unit ?**
multibus *busno* **on mbad** *interface* **unit ?**

DESCRIPTION

The MULTIBUS adapter (mbad) is a Sequent computer system board which provides a MULTIBUS interface for the Sequent computer. The mbad number is specified by *interface* in the configuration entry. The MULTIBUS number, *busno*, and the mbad interface number, *interface*, must be assigned in sequential order starting at zero. The unit number, **unit** , must be specified with "?" in both of the configuration entries shown above. The "?" is a wildcard that allows the autoconfiguration subsystem to assign correct unit numbers.

The MULTIBUS devices supported are the **lp**(7), Systech parallel line printer interface, **st**(7), Systech MTI-800/1600/1650 terminal multiplexor driver, and **xt**(7), Xylogics 472 MULTIBUS magtape interface.

SEE ALSO

lp(7), st(7), xt(7)

NAME

mem, kmem – core memory

CONFIGURATION ENTRY

pseudodevice mem

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

WARNINGS

Some of */dev/kmem* cannot be read because of write-only addresses or unequipped memory addresses.

NAME

mtio – UNIX magnetic tape interface

DESCRIPTION

Operating system magtape device names adhere to this naming convention:

/dev/{r}mt/<driver-name><unit><format>{n}

<driver-name> is either **ts** for SCED-based SCSI tape devices, **tm** for SSM-based SCSI tape devices, **tg** for SSM-based SCSI GCR tape devices or **xt** for MULTIBUS-based GCR tape devices. *<unit>* is a decimal number which corresponds to the specific tape drive. The value of *<format>* is defined for 1/2" GCR tape drives only. *<format>* may be **h, m** or **l** for tape densities of 6250, 1600 or 800 bpi, respectively. If density is not specified, the density currently set and showing on the tape drive's front panel is used. Finally, the suffix **n** added to the end of the device name indicates that the driver will not rewind the tape on close.

When a file open for writing is closed, two end-of-files are written. If the tape is not to be rewound, it is positioned with the head between the two tapemarks. If the most recent tape operation in not a valid tape ioctl, the tape is left at its current location.

A standard tape consists of a series of 1024-byte records terminated by an end-of-file. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it tends to create very large record gaps.

When foreign tapes are to be dealt with, and especially when long records are to be read or written, the 'raw' interface is appropriate. The associated files are named */dev/rmt/<driver-name><unit><format>{n}* but the same minor-device considerations as for the regular files still apply. A number of other ioctl operations are available on raw magnetic tape. Not all operations are supported by each driver. The following definitions are from *<sys/mtio.h>*:

```
/*
 * Structures and definitions for mag tape io control commands
 */

/*
/* structure for MTIOCTOP - */ mag tape op command */
*/
struct  mtop  {
  short mt_op;          /* operations defined below */
  daddr_t mt_count;     /* how many of them */
};

/*
/* operations */
*/
#define MTWEOF  0       /* write an end-of-file record */
#define MTFSF 1         /* forward space file */
#define MTBSF 2         /* backward space file */
```

```
#define MTFSR 3         /* forward space record */
#define MTBSR 4         /* backward space record */
#define MTREW 5         /* rewind */
#define MTOFFL  6       /* rewind and put the drive offline */
#define MTNOP 7         /* no operation, sets status only */
#define MTERASE 8       /* erase from current position to EOT */
#define MTRET 9         /* retension the (streamer) tape */
#define MTSEOD  10      /* skip to end of (streamer) tape data */
#define MTNORET 11      /* don't retention the tape this time */

/*
/* structure for MTIOCGET - mag tape get status comman
*/
struct  mtget {
  short mt_type;        /* type of magtape device */
/* the following two registers are device dependent */
  short mt_dsreg;       /* ''drive status'' register */
  short mt_erreg;       /* ''error'' register */
/* end device-dependent registers */
  short mt_resid;       /* residual count */
/* the following two are not yet implemented */
  daddr_t mt_fileno;    /* file number of current position */
  daddr_t mt_blkno;     /* block number of current position */
/* end not yet implemented */
};

/*
 * Constants for mt_type byte
 */
#define MT_ISTS   0x01
#define MT_ISHT   0x02
#define MT_ISTM   0x03
#define MT_ISMT   0x04
#define MT_ISUT   0x05
#define MT_ISCPC    0x06
#define MT_ISAR   0x07
#define MT_ISXT   0x08
#define MT_ISST   0x09
#define MT_ISTB   0x0A

/*
/* mag tape io control commands */
*/
#define MTIOCTOP  _IOW(m, 1, struct mtop) /* do a mag tape op */
#define MTIOCGET  _IOR(m, 2, struct mtget) /* get tape status */
```

Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size; if the record is long, an error is indicated. In raw

tape I/O seeks are ignored. A zero byte count is returned when a tape mark is read, but another read will fetch the first record of the new tape file.

FILES

/dev/mt/* and /dev/rmt/*

SEE ALSO

mt(1), tar(1), ts(7), tm(7), tg(7), xt(7)

BUGS

The status should be returned in a device independent format.

The auto density-select functionality for 1/2" gcr-type tape drives is not implemented in some drivers.

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

pmap – map physical addresses into process's virtual address space

CONFIGURATION ENTRY

pseudodevice pmap *number*

DESCRIPTION

The **pmap** device driver is used in conjunction with the **mmap**(2seq) system call to map arbitrary physical addresses into a process's virtual address space. The driver supports *number* minor devices, each of which may represent a specified region of the physical address space. The region associated with each device may be defined at kernel build time in the file */etc/conf/uts/io/pmap/pmap_space.c*, and can be altered at run time by the superuser via the **pmap**() utility or the ioctls described in this manual entry.

The **pmap** driver can support both physical devices (such as MULTIBUS address space) and non-paged memory devices (such as special areas of memory reserved for accelerators).

In */etc/conf/uts/io/pmap/pmap_space.c*, the boot-time parameters of the minor devices are defined as an array of struct **pmap_unit**, which is defined as follows:

```
struct pmap_unit {
  u_long            pm_paddr;     /* start physical (byte) address */
  u_long            pm_size;      /* size (in physical pages) */
  u_char            pm_flags;     /* flags (see below) */
  <other fields>                  /* not user visible, initially zero */
};
```

The *pm_paddr* field must be a multiple of the system page size (see **getpagesize**(2seq)), and the *pm_size* field must be a multiple of (system page size)/(hardware page size). The hardware page size is defined as NBPG in */usr/include/sys/mc_param.h*. There are three flags:

```
#define       PMAP_EXCL     0x01   /* exclusive access */
#define       PMAP_MAPPED   0x02   /* is currently mapped */
#define       PMAP_NPMEM    0x04   /* non-paged memory, else physical */
```

The PMAP_EXCL and PMAP_MAPPED flags should not be set in */etc/conf/uts/io/pmap/pmap_space.c*. PMAP_NPMEM should be set if the associated device is of the non-paged memory variety. However, these requirements are not checked.

There is no **read** or **write** system call interface to this driver.

IOCTLS

The **pmap** driver supports the following ioctls. Some ioctls use the following structure, which is defined in */usr/include/sys/pmap.h*:

```
structpmap_ioc {
  u_long    pi_paddr;    /* start physical (byte) address */
  u_long    pi_size;     /* size (bytes) */
  u_char    pi_flags;    /* flags (see above) */
};
```

Note that in the **pmap_ioc** structure, the size is specified in bytes, while in the **pmap_unit** structure it is specified in hardware pages.

```
ioctl(fd, PMAPIOCGETP, pmap_data)
int    fd;
structpmap_ioc *pmap_data;
```

Returns the current parameters from the **pmap_unit** table for the open unit referenced by *fd*. Any process can execute this call. EFAULT results for unreasonable values of pmap_data. The flags bits are as defined above: PMAP_MAPPED tells if the unit is currently mapped, PMAP_EXCL indicates that exclusive use is currently in effect, and PMAP_NPMEM indicates that *fd* is configured as a non-paged memory device.

```
ioctl(fd, PMAPIOCSETP, pmap_data)
int    fd;
structpmap_ioc *pmap_data;
```

PMAPIOCSETP allows a root process to set entries in the driver's **pmap_unit** table. EPERM results if the caller is not root. EFAULT results for unreasonable values of *pmap_data*. EBUSY results if the unit is mapped at the time of this call. EINVAL results if *pi_paddr* or *pi_size* is not a multiple of the system page size. The driver cannot verify the legality of the physical addresses passed in: user beware.

Only the PMAP_NPMEM flag bit is copied from *pi_flags* to *pm_flags*. Use PMAPIOCEXCL or PMAPIOCNXCL to set or clear the PMAP_EXCL bit.

NOTE: This ioctl allows the superuser to associate a file with any range of physical addresses, including MULTIBUS mapping registers, nonexistent memory, etc. Specifying incorrect values for *pi_paddr* or *pi_size* can lead to catastrophic results.

```
ioctl(fd, PMAPIOCEXCL, 0)
int    fd;
```

PMAPIOCEXCL verifies that there is at most one open at the time of the ioctl, and ensures that no further (non-superuser) opens are allowed on the unit until it is closed or the PMAPIOCNXCL ioctl is called. PMAPIOCEXCL fails (returning EBUSY) if there is already more than one open. Any process may execute this call. Note that the semantics are similar, but not identical, to TIOCEXCL in the terminal driver.

```
ioctl(fd, PMAPIOCNXCL, 0)
int    fd;
```

PMAPIOCNXCL disables exclusive use of the unit, thus allowing further opens. Any process may execute this call.

SEE ALSO

mmap(2seq), pmap(1M), getpagesize(2seq)

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 **sysadm** System Administration to access the devices on which it operates. Not all */dev/(r)SA* entries are used by all System Administration commands and not all possible devices will have links in these directories. If there are devices that you want supported from within **sysadm**, you need to create hard links from */dev/SA/* to them.

FILES

/dev/SA
/dev/rSA

SEE ALSO

sysadm(1)

NAME

SCED – SCSI/Ethernet/Diagnostics board

CONFIGURATION ENTRY

sec *ctrl* **on sb8000 unit ?**
sec_scsi *scsi_busno* **on sec** *ctrl* **unit ?**

DESCRIPTION

The Sequent SCSI/Ethernet/Diagnostics board is a Sequent computer system board which contains two console ports, an Ethernet interface and a SCSI bus interface. The SCED board number is specified by *ctrl* in the configuration entry. The SCSI bus number *scsi_busno* must be assigned to each bus in sequential order starting at zero. The SCED unit number, **unit** , must be specified with "?" in the two configuration entries shown above. The "?" is a wild-card that allows the autoconfiguration subsystem to assign correct unit numbers.

The SCED-based console ports are automatically configured for each configured SCED.

SEE ALSO

ts(7), console(7)

NAME

sd – SCSI disk controller

CONFIGURATION ENTRY

sd on sec_scsi *sec_scsi* **unit** *unit*

DESCRIPTION

This device driver controls both SCSI disk target adapters and SCSI disks with embedded targets ("embedded SCSI" disks). It supports up to eight disks per target adapter, with a maximum of seven target adapters per SCED board, or it can support a maximum of seven embedded SCSI disks per SCED board.

The SCSI disk is located on SCED SCSI bus *sec_scsi* and is configuration unit number *unit*. The configuration unit number is an index into the *sec_scsi_config* table in the file */etc/conf/uts/io/sec/sec_space.c*, which specifies the target adapter number, the logical unit number, and whether or not it is an embedded SCSI disk. The unit number must be unique with respect to other devices on this SCED SCSI bus. Both the *sec_scsi* and *unit* may be specified with a "?," meaning they are wild-carded. If *sec_scsi* is wild-carded, the system will assign the number of the SCED SCSI bus on which the disk is physically located. If *unit* is wild-carded, the system will assign the configuration unit number of the first unused matching entry in the *sec_scsi_config* table. A disk matches if its target adapter number matches the target adapter number specified in the *sec_scsi_config* table entry. Mixing of embedded SCSI and target adapters is not allowed on one system.

The special file with minor device number 255 refers to drive **sd0** (the first drive listed in the kernel configuration file), minor device number 511 refers to **sd1**, and so on. Standard device names are of the form **sd***n*, where *n* is the drive number.

Block devices access the disk via the system's normal buffering mechanism, and may be read and written without regard to physical disk records. These special files are usually found in */dev/dsk* — for example, */dev/dsk/sd0*. There is also a *raw* interface that provides for direct transmission between the disk and the user's read or write buffer. Raw transfers are limited to a maximum of 254 512-byte sectors. In raw I/O, counts must be a multiple of 512 bytes, and must begin on 16-byte aligned memory addresses. Likewise, **lseek** calls should specify a multiple of 512 bytes. A single read or write call usually results in exactly one I/O operation, which transfers data directly to the users address space. Therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw special files are conventionally the same as the block special files, and are found in the directory **/dev/rdsk** — e.g., **/dev/rdsk/sd0**.

NOTE: The sd driver provides access to the entire disk, including reserved sectors. To make the device usable, a partitioning driver (like **vtoc**(7) or **cmpt**(7)) must be built on the unpartitioned disk.

BINARY CONFIGURATION

Before attempting to add a new disk or target adapter, read the disk and target adapter manuals thoroughly. For this application, the most important information is how to set the target adapter number and drive-select cabling.

Table of disk units

The **sdbconf** array in the device driver binary configuration file (*/etc/conf/uts/io/sd/sd_space.c*) must contain an entry for each drive that may be configured; the entries must be in the same order as the "device" entries in the kernel configuration file. When adding a new drive, make sure that the corresponding entry in **sdbconf** points to the partition table for the appropriate drive type. (As installed, **sdbconf** has entries for 8 drives, all using the same partition table. Superfluous entries are ignored; you must add more entries if you have more than 8 drives.)

Tuning Parameters

The following tuning parameters are defined for each drive in the **sdbconf** array of the binary configuration file: (1) the buffer size for ioctl operations, (2) the number of IATs (indirect access table entries, used for scatter/gather I/O), and (3) water marks to lessen the interrupt load on the SCED board.

IATs

An indirect access table is used to map a contiguous I/O buffer in virtual memory to the corresponding pages scattered throughout physical memory. Each table entry (IAT) can map up to 2048 bytes. The number of IATs directly affects the total number of of outstanding requests that the device driver can queue to the SCED board. Groups of IATs are allocated on a per-transfer basis, and there must be enough entries to handle at least one request.

Water Marks

The water marks are **low** and **thresh**(hold). The device driver will queue up a maximum of **thresh** requests in a drive's device output queue before interrupting the SCED board to handle the requests. The **low** water mark is used when the device input queue has been filled and is draining off. The device driver does not begin queueing requests again until the "low" water mark has been reached. (The size of a drive's device input and output queues is defined by the "doneq" and "req" fields of the corresponding "device" entry in the kernel configuration file.)

FILES

/dev/dsk/sd?s?	Block files
/dev/rdsk/sd?s?	Raw files
/etc/conf/uts/io/sd/sd_space.c	Binary configuration file
/etc/conf/uts/io/sec/sec_space.c	SEC SCSI configuration table

SEE ALSO

devbuild(1M), mkvtoc(1M), newfs(1M), prtvtoc(1M), vtoc(7), cmpt(7), CCSformat(8)

DIAGNOSTICS

See the ANSI X3T9.2 (SCSI) specification.

BUGS

In raw I/O, **read**(2),**write**(2), and **lseek**(2) should always deal in 512-byte multiples.

A program to analyze the logged error information (even in its present reduced form) is needed. Error messages should be more consistent.

The kernel "milliseconds per seek" statistic is not kept for this device; thus, **iostat**(1) always reports zero for this statistic.

When a request to the disk accesses past the end of media (for example, a non-existent block number), the device driver will report soft errors followed by a hard error in the same way that a bad block is treated. There is no special handling of the volume overflow sense error code for disks.

NAME

sp – STREAMS pipe device

CONFIGURATION ENTRY

pseudodevice sp [*count*]

DESCRIPTION

The **sp** driver provides a machine-local interprocess communication mechanism using streams. The communication mechanism works by cross-connecting the "bottom" of two streams, so that writes to one stream can be read on the other. To facilitate selection of an unused streams pipe device, the **sp** device driver automatically allocates the next available master when invoked from the STREAMS clone device driver by opening **/dev/spx** (see **clone(7)**). An open file descriptor with read and write access to a streams pipe device is returned.

To use a streams pipe, the application must first open two streams pipe devices (either by explicit name or by clone open). Call the file descriptors returned from the opens *fd0* and *fd1*. Next, the application must "connect" the two streams pipes *(fd0* and *fd1)* using a *I_FDINSERT* ioctl (see **streamio(7)**).

Once two streams pipes are connected, a write on one "end" of the pipe can be read on the other end. A write of data onto *fd0* can be read on *fd1*, and vice-versa. This communication scheme differs from regular pipes in that it's "full duplex."

Streams pipes provide machine local, bidirectional communication with all the basic interfaces available to streams: the application can **poll()** on streams file descriptors, and it can use either byte-oriented (**read(), readv(), write(), writev()**) or record-oriented transfers (**getmsg()/putmsg()**).

Streams pipes have another facility, called "file descriptor passing." The mechanism uses *I_SENDFD,* and *I_RECVFD* **ioctl()'s** to pass an open file descriptor from one process to another through a streams pipe (see **streamio(7)**). With this mechanism, it's possible for unrelated processes to have pointers to the same file table entry. This can be used to create a client/server communication structure. The facility works like this:

fd0 and *fd1* are streams pipes which are opened and connected by the parent process of a pool of servers. The parent then does a **mknod()** system call to create a character special file on disk with the major/minor numbers of *fd1,* and giving it a name, for example *server_head. server_head* is now a globally accessible name for the *fd1* end of the server's streams pipe. Every member of the server pool will then do **ioctl(fd0, I_RECVFD, &fdstructure)**. Each server will block until it receives a file descriptor. This file descriptor will be from a client wanting to communicate with a server. This file descriptor may be for one end of another streams pipe pair, or it may be for an ordinary pipe or even a disk file. The server can then converse with the client through the passed file descriptor. When the conversation is finished, the server can close the passed fd and wait again at the **ioctl()**.

The client wanting to talk with a server first creates its own connected streams pipe pair, e.g., getting back *fda* and *fdb*. Then, it would open the special file *server_head* (see above) created by the first server, receiving a file descriptor *fdc*. The client can then perform the system call **ioctl(fdc, I_SENDFD, fda)** to send the file descriptor for one of the pipe ends for his

streams pipe to a server. The server and client can then communicate over the streams pipe created by the client, using file descriptors *fda* and *fdb*.

SEE ALSO

streams(7), clone(7), streamio(7)

NAME

SSM – System Services Module

CONFIGURATION ENTRY

ssm *ctrl* **on sb8000 unit ?**
ssm_scsi *scsi_busno* **on ssm** *ctrl* **unit ?**
vme *vme_busno* **on ssm** *ctrl* **unit ?**

DESCRIPTION

The System Services Module (SSM) is a Sequent computer system board that contains two console ports, a SCSI interface, a VMEbus interface and a parallel printer port interface. The SSM number is specified by *crtl* in the configuration entry. The SCSI bus number *scsi_busno*, and the VME bus number *vme_busno*, must be assigned in sequential order starting at zero. The SSM unit number, **unit** , must be specified with "?" in all three configuration entries shown above. The "?" is a wild-card that allows the autoconfiguration subsystem to assign correct unit numbers.

The SSM-based console ports and printer ports are automatically configured for each configured SSM.

SEE ALSO

tm(7), console(7), wd(7), wp(7)

NAME

st – Systech MTI-800/1600/1650 terminal multiplexor driver

CONFIGURATION ENTRY

st on multibus *multibus* **unit** *unit* **segsize 1024 nsegs 16**

DESCRIPTION

An MTI-800 provides 8 serial lines; an MTI-1600/1650 provides 16 serial lines. Each line of the MTI-800/1600/1650 is configurable to support a modem or a terminal.

The controller board is located on MULTIBUS *multibus* and is configuration unit number *unit*. The configuration unit number is an index into the *st_multibus_config* table in the file */etc/conf/uts/io/st/st_space.c*. The unit number must be unique with respect to other st devices on this MULTIBUS. Both the *multibus* and *unit* may be specified with a "?," meaning they are wild-carded. If *multibus* is wild-carded, the system will assign the number of the MULTIBUS in which the board is physically located. If *unit* is wild-carded, the system will assign the configuration unit number of the first unused matching entry in the *st_multibus_config* table. A board matches if it is at the address specified in the *st_multibus_config* table entry. The interrupt vector cannot be checked, so care must be taken to ensure that the interrupt vector is jumpered to match the *st_multibus_config* table entry. The *segsize* field is the size of address space reserved for DMA transfers and must be 1024. The *nsegs* field indicates how many regions to reserve for DMA transfers for this device and must be 16.

Each line attached to the MTI-800/1600/1650 terminal multiplexor behaves as described in **streamtty**(7). Input and output for each line may independently be set to run at any of the following speeds: **B50, B75, B110, B134, B150, B300, B600, B1200, B1800, B2400, B4800, B9600,** and **B19200**.

The hardware signals DTR and RTS are enabled when the line is opened and disabled when the line is closed if the HUPCL flag is set. These signals may be changed via ioctl calls to the device driver and are used when in input flow control is enabled. Break detection and generation are also supported. See **termio**(7) for the encoding scheme.

The initial baud rate and line state for all lines are defined by the *stcflags* variable in the binary configuration file, *st_space.c*. These values are usually overridden (e.g., by **getty**(1a)) when a line is opened.

Streams Driver Characteristics

The **st** driver is a STREAMS-based tty driver as described in **streamtty**(7). Functionality unique to this driver is described below:

When a set-parameters ioctl message is received with a baud rate of B0, this is considered to be a hangup message. DTR and RTS lines are dropped immediately.

If a set-parameters ioctl message is received with an illegal baud rate, the message will be rejected and the line state will not be changed.

The Systech hardware is not allowed to perform any flow control. The ld0 module and the driver execute all flow control processing. If the driver receives an M_PCCTL message with a sub-type of MC_STOPI or MC_STARTI, only the first data byte in the message is sent to the peripheral device.

If the driver receives an M_IOCTL/M_PCIOCTL message with a sub-type of TCGETA/TCGETP and the message does not arrive with a pre-allocated data buffer, the driver will allocate a streams buffer and attach it to the message.

Autodialer Support

For normal Systech operation, the DCD input must be active to allow the USART receiver to operate. A port can be *specially* wired to support an auto-dial modem (both dial-out *and* dial-in on the same line) without requiring the modem's DCD to be active. The special rewiring consists of the following steps.

1) Disconnect pins 4 (RTS), 8 (DCD), 6 (DSR) from the modem to the Sequent system.
2) Loop pin 4 (RTS) to pin 8 (DCD) on the backpanel board of the Sequent system.
3) Connect pin 8 (DCD) from the modem to pin 6 (DSR) on the Sequent system.

The following table illustrates the specially-wired connection.

Sequent system (Male conn.) Pin #		Terminal (Female conn.) Pin #	Description
2	to	2	TxD driven by Sequent system
3	to	3	RxD driven by Modem
5	to	5	CTS driven by Modem
6	to	8	DCD driven by Modem
7	to	7	Signal Ground
20	to	20	DTR driven by Sequent system

Pin 4 is looped back to Pin 8 on the Sequent system side.

A logic-high-to-logic-low transition on the auto-dialer's DCD output is interpreted as a hangup signal.

FILES

/dev/tty<subdir>/<basename> /etc/conf/uts/io/st/st_space.c

SEE ALSO

tty(7), streamtty(7)

BUGS

The Systech MTI-800/1600 board does not use a DMA input transfer scheme. Therefore, excessive overrun errors may result under heavy load.

NAME

streamio – STREAMS ioctl commands

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.

BINARY CONFIGURATION

Autopush is a mechanism for automatically pushing a module when a stream is first opened. The following structure is used to define the actions taken:

```
struct st_ap {
        char            strap_driver[FMNAMESZ+1];
        unsigned int    strap_lminor;
        unsigned int    strap_hminor;
        char            strap_module[FMNAMESZ+1];
        int             strap_next;
} str_ap[];
```

If a streams module whose name matches *strap_driver* is opened and the minor is within the range

strap_lminor ≤ minor ≤ strap_hminor

then "strap_module" is pushed. If "strap_next" is not equal to -1, it is used to index the table, and its "strap_module" is pushed. Each time a module is pushed, the corresponding entry is searched for in the table, and, if found the specified modules are pushed instead of the one named. Thus the driver specifies a list of modules to be pushed and each module specifies a list of components. The table is initialized in the kernel configuration file */etc/conf/uts/kernel/i386_space/autop_space.c*. If an error during the push occurs, the open fails.

IOCTLS

The terminal **ioctl**(2) system calls have the form:

```
#include <sys/types.h>
#include <streamio.h>  or  #include <streamios.h>

int ioctl (fildes, command, arg)
```

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:

- [EFAULT] *arg* points outside the allocated address space.
- [EINVAL] Invalid module name.
- [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:

- [EINVAL] Invalid *arg* value.
- [ENOSR] Unable to allocate buffers for flush message due to insufficient STREAMS memory resources.
- [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 bit-wise-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:

[EAGAIN] — Allocation of a data structure to store the signal request failed.

[EINVAL] — *arg* value is invalid or *arg* is zero and process is not registered to receive the SIGPOLL signal.

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:

[EFAULT] — *arg* points outside the allocated address space.

[EINVAL] — Process not registered to receive the SIGPOLL signal.

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 will only look 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:

[EBADMSG] Queued message to be read is not valid for I_PEEK.

[EFAULT] *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.

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.

[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*.

[ENOSR] Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.

[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:

[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.

[ENOSR] Unable to allocate buffers for the *ioctl* message due to insufficient STREAMS memory resources.

[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:

[EAGAIN] Temporarily unable to allocate storage to perform the I_LINK.

[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.

[ENOSR] Unable to allocate storage to perform the I_LINK due to insufficient STREAMS memory resources.

[ENXIO] Hangup received on *fildes*.

[ETIME] Time out before acknowledgement message was received at *stream head*.

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 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_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)

DIAGNOSTICS

Unless specified otherwise above, the return value from *ioctl* is 0 upon success and -1 upon failure with *errno* set as indicated.

NAME

streampty – STREAMS pseudo terminal driver

CONFIGURATION ENTRY

pseudodevice pty *count*

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 (also known as a *controller* device) and a *slave* device. The slave device provides processes an interface identical to that described in **termio**(7). However, whereas all other devices which provide the interface described in **termio**(7) have a hardware device of some sort behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, anything written on the master device is given to the slave device as input and anything written on the slave device is presented as input on the master device.

Usually only one process has the master device opened while several processes may have the slave device opened. An attempt to open a master device which is currently in use (already open) will fail with *errno* set to **[EIO]**. (However, an already open file descriptor for a master device can be duplicated via either **dup**(2) or **fork**(2).) To facilitate selection of an unused master device, the master device driver automatically allocates the next available master when invoked from the STREAMS clone device driver by opening */dev/getpty*. An open file descriptor to the selected master device is returned. The selected master/slave pty pair can be identified via a **TIOCPTYDEV ioctl**(2) call against the opened master. Typically, a process selects an available master device via **getpseudotty**(3SEQ) which automatically performs these steps.

Note that while it is not possible to open a master device which is aleady open, it is possible to open a master device which is closed but whose corresponding slave device is currently opened by one or more processes. To prevent these possible processes from interfering with future pty use, **fvhangup**(2SEQ) can be used on the slave device after opening it.

While the slave device supports the full set of **ioctl**(2) calls desribed in **termio**(7), the master device supports only the following set.

The master device **ioctl**(2) system calls have the form:

```
#include <sys/types.h>
#include <termio.h>
#include <sys/strpty.h>

int ioctl (fildes, command, arg)
```

Where *fildes* is an open master device file descriptor. *arg* is of different types depending on the *command*. Valid *command* values are:

TIOCGWINSZ Get the current **winsize** structure settings of the slave device. See the description in **termio**(7).

TIOCSWINSZ Set the current **winsize** structure settings of the slave device. See the description in **termio**(7).

TIOCPTYDEV Get the major and minor numbers of both the master and slave devices and store in the **ptydev** structure referenced by *arg*. This structure has the form:

```
struct       ptydev {
  unsigned intptym_major;  /* master major number */
  unsigned intptym_minor;  /* master minor number */
  unsigned intptys_major;  /* slave major number */
  unsigned intptys_minor;  /* slave minor number */
}
```

If a temporary lack of system resources prevents an **ioctl**() from being successfully executed, the **ioctl**() returns –1 with **errno** set to [EAGAIN]. In this case, the caller should retry the **ioctl**(), possibly after waiting a short period of time.

SLAVE DEVICE DRIVER

The slave device **pty** driver is a STREAMS driver that implements the **streamtty**(7) interface with the following exceptions:

Terminal State Parameters

All baud rate values in *c_cflag* are ignored.

Opening a Slave Device File

If the CLOCAL (*c_cflag*) mode is set, or if either the O_NONBLOCK or the O_NDELAY flag is specified in the **open**(2), the open will not wait for an open on the corresponding master device to be present before succeeding. All other *c_cflag* flag bits are ignored.

Controlling terminal allocation is performed as described in **streamtty**(7).

All other **termio**(7) functions must be performed by STREAMS modules pushed on top of the slave driver; these are typically performed by **ld0**(7).

Closing a Slave Device File

The last process to close the slave device will cause the driver close routine to be called. This, in turn, will send an **M_ERROR** message up the master device stream causing all subsequent access to the master device by an application to fail with **errno** set to **[EIO]**.

General STREAMS Concerns

M_FLUSH message processing is handled somewhat differently than a typical STREAMS driver since a pty driver is not a logical endpoint for the stream. After performing the requested flush operation, rather than turning around the **M_FLUSH** message and sending it upstream (or else consuming it), the slave (master) driver propogates the message to the read-side of the corresponding master (slave) device stream. However, a FLUSHR is translated to a FLUSHW and vice versa. The master (slave) device driver performs the translated operation before forwarding the message up its stream. The stream head for the master (slave) device turns around the M_FLUSH if appropriate. Similarly, **ioctl**(2) commands such as TCSETAF and TCSETPF which require flushing not only flush the slave device stream but also the master device stream.

Slave Read-Side Generated Messages

Read-side-generated messages defined in **streamtty**(7) are supported with the

following changes:

All data written to the corresponding master device is sent upstream in M_DATA messages from the slave device.

Closing the corresponding master device is treated as a line disconnect for the purposes of generating an **M_HANGUP** message (see **streamtty**(7)).

M_COPYIN and **M_COPYOUT** messages are generated in response to a TRANSPARENT M_IOCTL message.

The following messages are not generated: **M_CTL, M_PCCTL.**

Slave Write-Side Received Messages

Write-side received messages defined in **streamtty**(7) are supported with the following changes:

All data received in **M_DATA** messages is sent upstream in **M_DATA** messages from the corresponding master device.

M_IOCDATA messages are recognized if they apply to a TRANSPARENT **ioctl** being currently performed.

The **TCSBRK M_IOCTL** is supported although no break is generated.

The **TIOCGWINSZ** and **TIOCSWINSZ M_IOCTL** is supported in the same manner as ld0(7). This replaces the processing **ld0** would do for these **ioctl**s and is necessary in order to allow access to the information from the master device driver.

IXOFF and **IXON** input and output flow control via the **MC_STOP, MC_STOPI, MC_START,** and **MC_STARTI** forms of the **M_PCCTL** message are not supported directly by the pty driver. Input flow control is implicitly provided by the STREAMS mechanism since there is no hardware device to flow control. **IXON** output flow control via the stop and start characters can still be performed when the **ld0**(7) module is pushed ontop of the slave device driver.

The following messages are ignored: **M_DELAY, M_PCCTL.**

MASTER DEVICE DRIVER

The master device **pty** driver is a STREAMS driver which connects with the corresponding slave device **pty** driver. It does *not* support the **termio**(7) interface. There should be no additional STREAMS modules pushed on top of the master device driver.

Opening a Master Device File

Typically, a master device is opened via a clone open which selects the next available (unused) master device and opens that. The identity of the selected master/slave pty pair can be retrieved with a subsequent TIOCPTYDEV **ioctl**(2) call. If no unused master device is currently available, the open fails with *errno* set to **[EBUSY]**.

Alternately, a particular master device may be explicitly opened. An open of a particular master device will fail (with **errno** set to **[EIO]**) if that master device is currently already open. A master device cannot become a controlling terminal; a slave device does this.

Closing a Master Device File

The last process to close the master device will cause the driver close routine to be called. If the CLOCAL (*c_cflag*) mode is not set, an **M_HANGUP** message

will be sent up the slave device stream causing all subsequent access to the slave device by an application to fail in the same manner as if modem disconnect had been sensed on a terminal. (See **termio**(7).)

Master Read-Side Generated Messages

M_DATA All data received on the write-side of the corresponding slave device is sent upstream in M_DATA messages.

M_IOCACK Generated in response to any successfully executed M_IOCTL or M_PCIOCTL message.

M_IOCNAK Generated in response to any unsuccessfully executed or unrecognized M_IOCTL or M_PCIOCTL message.

M_FLUSH Generated in response to an M_FLUSH message received on the write-side of the corresponding slave device. A FLUSHR is translated to a FLUSHW and vice versa from the slave to the master side. The master device driver performs the translated operation before forwarding the message upstream.

M_COPYIN
M_COPYOUT
Generated in response to a TRANSPARENT M_IOCTL message.

Master Write-Side Received Messages

M_DATA The message is passed upstream on the corresponding slave device stream.

M_IOCTL
M_PCIOCTL
The supported **ioctl**(2) calls are described in the Master IOCTLS section below.

M_IOCDATA
Recognized if it applies to a TRANSPARENT **ioctl** being currently performed.

M_FLUSH If FLUSHW or FLUSHRW is specified, the write queue is flushed. The original M_FLUSH message is passed to the corresponding slave device driver read-side however a FLUSHR is translated to a FLUSHW and vice versa.

Master IOCTLS

The following **ioctl**(2) commands are supported as defined in **termio**(7).

TIOCPTYDEV M_IOCTL recognized during master write-side processing. Returns the major and minor numbers of both the master and slave devices. The information is returned as described above.

Applications should not use this; rather, **getpseudotty**(3SEQ) should be used instead.

TIOCGWINSZ TRANSPARENT M_IOCTL recognized during master write-side processing. Return in struct **winsize** the current setting of the window size of the corresponding slave device.

TIOCSWINSZ TRANSPARENT M_IOCTL recognized during master write-side processing. Sets the window size of the corresponding slave device based on the information provided in the struct

winsize. If any of the current settings are changed, an **M_SIG** message is sent up the slave device stream to generate a **SIGWINCH**.

FILES

/dev/getpty	Master pty clone device
/dev/tty??/????	Slave pseudo terminals

SEE ALSO

getpseudotty(3SEQ), ld0(7), streamtty(7)

BUGS

It is not possible to send an EOT.

NAME

streamtty – Generic STREAMS tty driver interface

DESCRIPTION

This describes a generic interface supported by a STREAMS tty driver that is designed to be used in conjunction with the **ld0** tty line discipline STREAMS module to implement the terminal interface described in **termio**(7). This does not describe any particular STREAMS tty driver but is rather an interface specification to which specific drivers should adhere. A specific driver may extend or, in some cases, implement less than what is documented here. Refer to the individual manual pages for information about a specific tty driver.

Terminal State Parameters

Several **ioctl**(2) calls described below get and set terminal state parameters as defined in **termio**(7). Note that some *ioctls* pass these parameters in a **termio** structure while some pass them in a **termios** structure; the driver is responsible for correctly interpreting the *ioctl* data as the appropriate structure.

Most of the parameters are relevant only to **ld0**. The following terminal parameters are relevant to the terminal driver.

The CBAUD (*c_cflag*) bits define the baud rate at which the terminal device operates. The valid baud rate values are described in **termio**(7). Unsupported baud rate changes are ignored. The B0 baud rate is interpreted as a line disconnect resulting in a hangup operation.

The CSIZE, CSTOPB, CLOCAL, PARENB, PARODD, and HUPCL (*c_cflag*) bits are supported if meaningful for the hardware.

Other parameters, while implemented by **ld0**, may also be interpreted by the driver in an effort to provide more intelligent or timely behavior as long as this behavior adheres to the **termio**(7) interface. For example: If IXON (*c_iflag*) is set, the driver could do output flow control (e.g., in hardware) using the START and STOP characters defined in the *c_cc* array rather than relying on **ld0** to send M_PCCTL messages requesting the MC_START and MC_STOP functions downstream in response to the START and STOP characters; IXANY (*c_iflag*) could be similarly handled. If CREAD (*c_cflag*) is not set, the terminal receiver hardware could be disabled rather than relying on **ld0** to silently discard input in this case.

Although **ld0** performs the TIOCGWINSZ and TIOCSWINSZ **ioctl**(2) calls, it does so only after the driver has had a chance to perform the *ioctl* itself. If the driver does not perform the *ioctl* but generates an M_IOCNAK message instead, **ld0** intercepts the M_IOCNAK and performs the *ioctl* itself.

Opening a Terminal Device File

When a terminal file is opened, it normally causes the process to wait until a connection is established. However, if the CLOCAL (*c_cflag*) mode is set in the terminal, or if either the O_NONBLOCK or the O_NDELAY flag is specified in the **open**(2), then the open will not wait for a connection to be established.

When a terminal file is opened for the first time (it is not already currently open), the driver-internal terminal state parameters are set to the appropriate default values as defined in **termio**(7).

A terminal may belong to a process as its *controlling terminal*. Each process of a session that has a controlling terminal has the same controlling terminal. A terminal may be the controlling terminal for at most one session. During open, a controlling terminal for a session is allocated by the session leader process in the following manner. If a session leader has no controlling terminal, and opens a terminal device file that is not already associated with a session without using the O_NOCTTY **open**(2) option, the terminal becomes the controlling terminal of the session leader and the session leader becomes the *controlling process* for that terminal. When a controlling terminal becomes associated with a session, its foreground process group is set to the process group of the session leader. This will also cause the stream head to use this process group as the destination for future signals generated by M_SIG and M_PCSIG messages. The session structure has a pointer to a "struct ctty" that is set, if the above conditions are met, to "sd_ctty" in the stream head. This should only happen for streams to a streamtty device whoes open routine indicates that it can become a controlling terminal.

The typical driver coding sequence is:

```
REQ_CTTY(); /* inform stream-head that */
            /* we can become a ctty    */
```

When a controlling process terminates, the controlling terminal is disassociated from the current session, allowing it to be re-acquired by a new session leader. The **exit**(2) function does this by zeroing the vnode pointer in the session structure for the process and the session pointer in the stream head. Subsequent access to the terminal by other processes in the earlier session are denied, with attempts to access the terminal treated at the stream head as if modem disconnect had been sensed. The routines ctty_read() and ctty_write() will fail because, although the session ID and ctty pointer match, the vnode pointer has been zeroed.

Closing a Terminal Device File

The last process to close a terminal device file will cause the driver close routine to be called. If there is no current active terminal connection, the write queue is flushed. If there is a current active terminal connection, then providing the driver flags have not been set to O_FNDELAY then the driver will perform an implicit M_PCCTL MC_START message (see below) and wait for any output data to finish being sent to the device. The driver will timeout this drain to guarantee that the close completes. If the timeout expires before all output data has been sent to the device, the remaining output data will be flushed.

Read-side Generated Messages

M_DATA All input data received from the device is immediately sent upstream in M_DATA messages with the exception of framing and parity errors which are treated as described below.

M_CTL The first data byte of the control message indicates the type of operation. Additional data bytes may be supplied dependent on the operation. Supported operations are:

MC_FRAME_ERR A framing error was detected. If the data value of the character with the framing error is known, it is provided in the remaining data bytes of the message. If not supplied, the data value of the character is assumed to be zero. A break character is represented as a framing error character whose data value is zero.

MC_PARITY_ERR
A parity error was detected. If the data value of the character with the parity error is known, it is provided in the remaining data bytes of the message. If not supplied, the data value of the character is assumed to be zero.

M_PCCTL The first data byte of the control message indicates the type of operation. Additional data bytes may be supplied dependent on the operation. Supported operations are:

MC_STOPI The driver requests that all input data messages stop if IXOFF is set. Typically, a **MC_STOPI** message is sent back to the driver.

M_IOCACK Generated in response to any successfully executed M_IOCTL or M_PCIOCTL message.

M_IOCNAK Generated in response to any unsuccessfully executed or unrecognized M_IOCTL or M_PCIOCTL message.

M_FLUSH Generated in response to either an M_FLUSH message with FLUSHR or FLUSHRW set, or a TCSETAF or TCSETPF M_IOCTL message.

M_HANGUP Request the stream head to perform a hangup operation. Generated in response to a line disconnect as a result of loss of modem control signals while CLOCAL (*c_cflag*) is not set.

Write-side Received Messages

M_DATA The data is delivered to the device for output.

M_DELAY Request a real-time delay on output. The data buffer contains an integer indicating the number of machine ticks of delay.

M_IOCTL
M_PCIOCTL
The *ioctl*(2) calls described in *termio*(7) that are supported by the tty driver are described in the IOCTLS section below.

M_PCCTL The first data byte of the priority control message indicates the type of operation. Additional data bytes may be supplied dependent on the operation. Supported operations are:

MC_STOP Suspend the flow of data from the driver to the device.

MC_START Resume the flow of data from the driver to the device.

MC_STOPI Suspend the flow of data from the device to the driver. The remaining data bytes contain the current STOP character.

MC_STARTI Resume the flow of data from the device to the driver. The remaining data bytes contain the current START character.

M_FLUSH If FLUSHW or FLUSHRW is specified, the write queue is flushed along with all data currently being output to the device (if possible). If FLUSHR or FLUSHRW is specified, the driver converts it to FLUSHR and sends the M_FLUSH message upstream.

IOCTLS

The following **ioctl**(2) commands are supported as defined in *termio*(7).

TCSBRK M_IOCTL recognized during write-side processing. Wait for output to drain, then optionally send a break (as described in **termio**(7)).

TCGETA
TCGETP M_PCIOCTL recognized during write-side processing. The message should already contain a data buffer sufficiently large to contain the necessary returned parameter structure (allocated above the stream head). The data buffer is interpreted to be the appropriate data structure type, the data structure is zeroed, the parameters relevant to driver operation (see above) are stored in the data structure, and the message is converted to an M_IOCACK message and passed upstream.

TCSETP
TCSETPW
TCSETPF
TCSETA
TCSETAW
TCSETAF Recognized during write-side processing. In the case of TCSETP and TCSETA the message is an M_PCIOCTL otherwise the message is an M_IOCTL. In the case of TCSETP or TCSETA an M_FLUSH message with FLUSHR set is passed upstream. In all cases the parameters relevant to driver operation (see above) are internally updated from the values in the data buffer of the received message and the message is converted to an M_IOCACK message and passed upstream.

SEE ALSO

stty(1), close(2), ioctl(2), open(2), ld0(7), termio(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.

All simultaneously open files that refer to the same terminal device share a common state for that terminal device. The input and output streams for the terminal device are shared and the parameters and state described below are also shared in common. This means, for example, if one process adjusts the parameters that effect input processing, all processes sharing the same terminal device are affected.

Opening a Terminal Device File

When a terminal file is opened, it normally causes the process to wait until a connection is established. However, if the CLOCAL mode is set in the terminal, or if either the O_NONBLOCK or the O_NDELAY flag is specified in the **open**(2), then open will not wait for a connection to be 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 Controlling Terminal

A terminal may belong to a process as its *controlling terminal*. Each process of a session that has a controlling terminal has the same controlling terminal. A terminal may be the controlling terminal for at most one session. The controlling terminal for a session is allocated by the session leader process in the following manner. If a session leader has no controlling terminal, and opens a terminal device file that is not already associated with a session without using the O_NOCTTY **open**(2) option, the terminal becomes the controlling terminal of the session leader and the session leader becomes the *controlling process* for that terminal. When a controlling terminal becomes associated with a session, its foreground process group is set to the process group of the session leader.

The controlling terminal is inherited by a child process during a **fork**(2). A process relinquishes its controlling terminal when it creates a new session with the **setsid**(2) or **setpgrp**(2) functions, or when all file descriptors associated with the controlling terminal have been closed.

When a controlling process terminates, the controlling terminal is disassociated from the current session, allowing it to be reacquired by a new session leader. Subsequent access to the terminal by other processes in the earlier session may be subsequently denied, with attempts to access the terminal treated as if modem disconnect had been sensed.

Process Groups

A terminal may have a foreground process group associated with it. This foreground process group plays a special role in handling signal-generating input characters, as discussed below. Also, command interpreter processes supporting 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. The terminal interface aids in the allocation by restricting access to the terminal by processes that are not in the current

foreground process group. This is accomplished in the following way:

Terminal Access Control

Attempts by a process in a background process group to read from its controlling terminal cause its process group to be sent a SIGTTIN signal unless one of the following special cases apply: If the reading process is ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is orphaned, the **read**(2) returns –1 with *errno* set to [EIO] and no signal is sent. The default action of the SIGTTIN signal is to stop the process to which it is sent.

Attempts by a process in a background process group to write to its controlling terminal cause its process group to be sent a SIGTTOU signal unless one of the following special cases apply: If TOSTOP (see the discussion of *c_lflag* below) is not set, or if TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, then the process is allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU, the **write**(2) returns –1 with *errno* set to [EIO] and no signal is sent.

Certain **ioctl**() calls that set terminal parameters are treated in the same fashion as write, except that TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set.

Input Processing and Reading Data

A terminal 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 at least 256 characters. When the input limit is reached, the buffer is flushed and all the saved characters are thrown away without notice.

Two general kinds of input processing are available, determined by whether the terminal device file is in canonical mode (ICANON set) or non-canonical mode. Additionally, input characters are processed according to the *c_iflag* and *c_lflag* fields. Such processing can include *echoing*, which means that input characters are put in the output queue as they arrive. This is useful for terminals that can operate in full-duplex mode.

Also the setting of the O_NONBLOCK and O_NDELAY flags via **open**(2) or **fcntl**(2) affect the operation of **read**(2). See the discussion of these flags for STREAMS devices in **read**(2). With respect to that discussion, note that when data is available depends on whether the input processing mode is canonical or non-canonical.

Canonical Mode Input Processing

In canonical mode, terminal input is processed in units of lines. A line is delimited by a new-line (NL) character, an end-of-file (EOF) character, or an end-of-line (EOL) 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.

Erase and kill processing occur when either of the two special characters, ERASE and KILL, is received. The ERASE character erases the last character typed, except that it will not erase beyond the beginning of the line. The KILL character kills (deletes) the entire input line, and optionally outputs a newline character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Under some modes (see IEXTEN), the ERASE, KILL, and EOF characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read.

Non-Canonical Mode Input Processing

In non-canonical mode input processing, input characters are not assembled into lines, and erase and kill processing does not occur. The values of the MIN and TIME members of the *c_cc* array are used to determine how to process the characters received.

MIN represents the minimum number of characters that should be received when the **read**() successfully returns. TIME is a timer of 0.1 second granularity that is used to timeout burst and short-term data transmissions. The four possible values for MIN and TIME and their interactions are described below.

Case A: 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. The interaction between MIN and TIME is as follows: as soon as one character is received, the intercharacter timer is started. If MIN characters are received before the intercharacter timer expires (remember that the timer is reset upon receipt of each character), the read is satisfied. If the timer expires before MIN characters are received, the characters received to that point are returned to the user. Note that if TIME expires, at least one character will be returned because the timer would not have been enabled unless a character was received. In this case (MIN > 0, TIME > 0) the read will block until the MIN and TIME mechanisms are activated by the receipt of the first character, or a signal is received.

Case B: MIN > 0, TIME = 0

In this case, since the value of TIME is zero, the timer plays no role and only MIN is significant. A pending read is not satisfied until at least MIN characters are received (the pending read will block until at least MIN characters are received), or a signal is received. A program that uses this case to read record-based terminal I/O may block indefinitely in the read operation.

Case C: MIN = 0, TIME > 0

In this case, since the value of MIN is zero, TIME no longer represents an intercharacter timer. It now serves as a read timer that is activated as soon as a **read**() is processed. A read is satisfied as soon as a single character is received or the read timer expires. Note that in this case if the timer expires, no characters 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 the read 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 a value of zero, having read no data.

Case D: 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. If no characters are available, **read**() will return a value of zero, having read no data. In this case, the **read**() never blocks, return is immediate.

Writing Data and Output Processing

When one or more characters are written, they are processed according to the *c_oflag* field and then transmitted to the terminal as soon as previously-written characters have finished typing. 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.

Also the setting of the O_NONBLOCK and O_NDELAY flags via **open**(2) or **fcntl**(2) affect the operation of **write**(2). See the discussion of these flags for STREAMS devices in **write**(2).

Special Characters

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII DEL) If ISIG is set, generates an *interrupt* signal (SIGINT) which is sent to all processes in the foreground process group associated with the 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). If NOFLSH is not set, the input and output queues are flushed when the signal is sent. If ISIG is set, the INTR character is discarded when processed.

QUIT (Control-| or ASCII FS) If ISIG is set, generates a *quit* signal (SIGQUIT). 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. If NOFLSH is not set, the input and output queues are flushed when the signal is sent. If ISIG is set, the QUIT character is discarded when processed.

SUSP (Control-z or ASCII SUB) If ISIG is set, generates a *terminal stop* signal (SIGTSTP) which is sent to all processes in the foreground process group associated with the terminal. Normally, each such process is forced to suspend, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see **signal**(2). If NOFLSH is not set, the input and output queues are flushed when the signal is sent. If ISIG is set, the SUSP character is discarded when processed.

ERASE (#) If ICANON is set, erases the preceding input character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character. If ICANON is set, the ERASE character is discarded when processed.

KILL (@) If ICANON is set, deletes the entire line, as delimited by a NL, EOF, or EOL character. If ICANON is set, the KILL character is

discarded when processed.

EOF (Control-d or ASCII EOT) If ICANON is set, 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 recognized if ICANON is set. It can not be changed.

EOL (ASCII NUL) is an additional line delimiter, like NL. It is not normally used.

EOL2 (ASCII NUL) is another additional line delimiter.

STOP (Control-s or ASCII DC3) If the IXON flag is set, can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. If the IXON flag is set, the STOP character is discarded when processed. While output is suspended, STOP characters are ignored and discarded. Also used for input flow control if the IXOFF flag is set.

START (Control-q or ASCII DC1) If the IXON flag is set, is used to resume output which has been suspended by a STOP character. If the IXON flag is set, the START character is discarded when processed. While output is not suspended, START characters are ignored and discarded. Also used for input flow control if the IXOFF flag is set.

CR (ASCII CR) the carriage return character. It can not be changed.

The character values for INTR, QUIT, SUSP, ERASE, KILL, EOF, EOL, EOL2, START, and STOP may be changed to suit individual tastes. Under some modes (see IEXTEN), the ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

If two or more special characters have the same value, the function performed when that character is received is undefined.

Modem Disconnect

If a modem disconnect is detected by the terminal interface for a controlling terminal, and if CLOCAL is not set for the terminal, the *hang-up* signal (SIGHUP) is sent to the controlling process associated with the terminal. Unless other arrangements have been made, this causes the controlling process to terminate. Any subsequent read from the terminal device by any process returns with an end-of-file indications until the device is closed. Thus, processes that read a terminal file and test for end-of-file can terminate appropriately after a disconnect. Any subsequent write to the terminal device returns –1, with *errno* set to [EIO], until the device is closed.

Closing a Terminal Device File

The last process to close a terminal device file will cause any output to be sent to the device and any input to be discarded. Then, if HUPCL is set, and the communications port supports a disconnect function, the terminal device shall perform a disconnect.

Terminal Parameters

Several **ioctl**(2) system calls apply to terminal files. Some of the calls use a structure which represents a collection of terminal state parameters. The structure is used to retrieve and to set terminal parameters. It should be noted that it is the invoking of the *ioctl*() calls, and not merely storing values into the structure, which affects the actual terminal state. There are two slightly different versions of the terminal state structure namely **termios** and **termio** and a third, obsolescent version, **sgttyb** that are used.

Termio and termios Terminal Parameters

The primary calls use the **termios** structure, defined in **<termios.h>**:

```
struct termios {
        tcflag_t    c_iflag;    /* input modes */
        tcflag_t    c_oflag;    /* output modes */
        tcflag_t    c_cflag;    /* control modes */
        tcflag_t    c_lflag;    /* local modes */
        cc_t        c_cc[NCCS]; /* 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 / VMIN    EOT
5    VEOL / VTIME   NUL
6    VEOL2          NUL
7    reserved
8    VSTART         DC1
9    VSTOP          DC3
10   VSUSP          SUB
```

The operating system **termio** structure is used by other **ioctl**(2) system calls; it is 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 / VMIN     EOT
5   VEOL / VTIME    NUL
6   VEOL2           NUL
7   reserved
```

The calls that use the **termio** structure only affect the flags and control characters that can be stored in the **termio** structure; all other flags and control characters are unaffected.

Input Modes

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 uppercase to lowercase 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, that is, not put on the input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with framing and parity errors (other than break) are ignored.

If PARMRK is set, a character with a framing or parity error (other than break) which is not ignored is read as the three-character sequence: '\377', '\0', X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of '\377' is read as '\377', '\377'. If PARMRK is not set, a framing or parity error (other than break) 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. Note that whether input parity checking is enabled or disabled is independent of whether parity detection (PARENB) is enabled or disabled. If parity detection is enabled (PARENB set) but parity checking is disabled (INPCK not set), the hardware to which the terminal is connected will recognize the parity bit, but the terminal special file will not check whether this bit is set correctly or not.

If ISTRIP is set, valid input characters are first stripped to 7 bits, otherwise all 8 bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). If IGNCR is not set and ICRNL is set, a received CR character is translated into a 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. When IXON is set, START and STOP characters are not read, but merely perform flow control functions. When IXON is not set, the START and STOP characters are read. If IXANY is set, any input character, will restart output which has been suspended and, if it is neither the start nor stop character, the character will still be read.

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 except the console which has ICRNL, IXON, IXOFF, BRKINT and IGNPAR set.

Output Modes

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 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 ('\177'), 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 except the console which has OPOST, ONLCR and TAB3 set.

Control Modes

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   reserved
XMT1EN      0020000   reserved
LOBLK       0040000   reserved
```

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. When directly manipulating the CBAUD bits in the *c_cflag* field, setting only one baud rate is possible which is interpreted to be both the input and output baud rate.

The following POSIX-conforming interfaces are also provided for getting and setting the values of the input and output baud rates in the **termios** structure. The effects on the terminal device do not become effective until the **tcsetattr()** function is successfully called.

```
#include <termios.h>

speed_t cfgetospeed (termios_p)
struct termios *termios_p;

int cfsetospeed (termios_p, speed)
struct termios *termios_p;
speed_t speed;
```

```
speed_t cfgetispeed (termios_p)
struct termios *termios_p;

int cfsetispeed (termios_p, speed)
struct termios *termios_p;
speed_t speed;
```

The type *speed_t* is defined in **<termios.h>** and is an unsigned integral type. It can assume any value defined above for the CBAUD bits.

The *termios_p* argument is a pointer to a **termios** structure. Note that use of a **termio** structure is unsupported with these POSIX-conforming interfaces.

cfgetospeed() returns the output baud rate stored in the **termios** structure pointed to by *termios_p*.

cfsetospeed() sets the output baud rate stored in the **termios** structure pointed to by *termios_p* to *speed*. The zero baud rate, B0, is used to terminate the connection. If B0 is specified, the modem control lines will no longer be asserted. Normally, this will disconnect the line.

cfgetispeed() returns the input baud rate stored in the **termios** structure.

cfsetispeed() sets the input baud rate stored in the **termios** structure to *speed*. If the input baud rate is set to zero, the input baud rate will be specified by the value of the output baud rate. Both **cfsetispeed**() and **cfsetospeed**() return a value of zero if successful and –1 to indicate an error. Attempts to set unsupported baud rates are ignored and no error is returned. This refers both to changes to baud rates not supported by the hardware, and to changes setting the input and output baud rates to different values if the hardware does not support this.

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.

Under normal circumstances, a call to **open**(2) will wait for the modem connection to complete. However, if the CLOCAL mode is set in the terminal, or if either the O_NONBLOCK or the O_NDELAY flag is specified in the *open*(2), then open will not wait for a connection to be established.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

Local Modes

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (*c_line* set to zero) provides the following:

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, quit, or suspend.
TOSTOP	0000400	Send SIGTTOU signal for background output.
IEXTEN	0200000	Enable extended functions.

If ISIG is set, each input character is checked against the special control characters INTR, QUIT, and SUSP. 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., '\377').

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, EOL, and, if IEXTEN is set, EOL2. See the Canonical Mode Input Processing section above for more details. 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. (These positions may also be referenced as VMIN and VTIME, respectively.) The time value represents tenths of seconds. See the Non-Canonical Mode Input Processing section above for more details.

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, when the ERASE character is processed it is echoed as ASCII BS SP BS if there is an input character to erase, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, when the ERASE character is processed it 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. 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). When processed, 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, interrupt, and suspend characters will not be done.

If TOSTOP is set, the SIGTTOU signal is sent to the process group of a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal, by default, stops the members of the process group. If TOSTOP is not set, or if the process writing to its controlling terminal is blocking or ignoring SIGTTOU signals, the output is allowed and the SIGTTOU signal is not sent. See the Terminal Access Control section above for more details.

IEXTEN is supported only for functions using the **termios** structure. Functions using the **termio** structure behave as if IEXTEN were set. If IEXTEN is set, the following extensions to the POSIX-compatible terminal interface are honored: XCASE in *c_lflag*, IUCLC and IXANY in *c_iflag*, the EOL2 special character, and escaping of the ERASE, KILL, and EOF special characters. If IEXTEN is not set, the POSIX-compatible terminal interface is strictly adhered to and the functional extensions listed above are disabled.

The initial line-discipline control value is the IEXTEN bit set and all other bits clear except the console which has in addition ISIG, ICANON, ECHO and ECHOK set.

The initial line-discipline selector value, *c_line* is zero. If an attempt is made to change *c_line* from this value, [EINVAL], is returned.

Sgttyb Terminal Parameters

The third, obsolescent version of the terminal state structure is struct **sgttyb** defined in **<ttold.h>**:

```
struct  sgttyb {
        char    sg_ispeed;
        char    sg_ospeed;
        char    sg_erase;
        char    sg_kill;
        char    sg_flags;
};
```

The functions **stty**(3) and **gtty**(3) use this structure as an argument which in turn is converted into an **ioctl**() TIOCSETP and TIOCGETP respectively.

The *sg_flags* field describes the terminal control:

```
O_HUPCL     0000001   Hang up on last close.
O_XTABS     0000002   Expand tabs to spaces on output.
O_LCASE     0000004   Canonical upper/lower presentation.
O_ECHO      0000010   Enable echo.
O_CRMOD     0000020   Canonical CR to NL mapping.
O_RAW       0000040   Canonical input processing.
O_ODDP      0000100   Odd parity, or with O_EVENP no parity.
O_EVENP     0000200   Even parity, or with O_ODD no parity
O_NLDELAY   0001400   Select new-line delays:
O_NL1       0000400   Termio CR1 delay if O_CRMOD set.
O_NL2       0001000   Termio CR2 delay if O_CRMOD set.
O_TBDELAY   0002000   Select horizontal-tab delays.
O_NOAL      0004000   Echo NL after kill character.
O_CRDELAY   0030000   Select carriage-return delays:
O_CR1       0010000   Termio CR1 delay.
O_CR2       0020000   Termio CR2 delay.
O_VTDELAY   0040000   Select vertical-tab delays.
O_BSDELAY   0100000   Select backspace delays.
```

IOCTLS

The **ioctl**() calls supported are listed below. Note that some calls operate on a **termios** structure while others operate on a **termio** structure. Some calls may not be supported by all devices or modules.

Unless otherwise noted for a specific **ioctl**() call, these functions are restricted from use by processes belonging to a background process group. Attempts to perform these operations shall cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation and the SIGTTOU signal is not sent. See the Terminal Access Control section above for more details.

The terminal **ioctl**(2) system calls have the form:

```
#include <sys/types.h>
#include <termio.h>  or  #include <termios.h>

int ioctl (fildes, command, arg)
```

In all the calls, *fildes* is an open file descriptor. (However, the calls affect the underlying terminal file, and not just the open file description associated with the file descriptor.) *arg* is of different types depending on the *command*. Upon successful completion, the **ioctl**() will return a value of zero. Otherwise, a value of –1 is returned and *errno* is set to indicate the error. The following *errno* values may be returned by any of the commands:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINVAL] Either *arg* or the data it references is not a proper value.

[ENOTTY] The file associated with *fildes* is not a terminal.

In addition to these, each command may use additional *errno* values unique to the specific command; these are described below.

The commands are:

TCGETP — Get the parameters associated with the terminal and store in the **termios** structure referenced by *arg*. Only those parameters that can be stored in the **termios** structure are returned. This command is allowed from a process in a background process group; however, the terminal attributes may be subsequently changed by a process in the foreground process group.

TCSETP — Set the parameters associated with the terminal from the **termios** structure referenced by *arg*. Only those terminal parameters that can be stored in the **termios** structure are set. The change is immediate.

TCSETPW — Same as TCSETP except that it first waits for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.

TCSETPF — Same as TCSETP except that it first waits for the output to drain and then flushes the input queue before setting the new parameters.

TCGETA — Get the parameters associated with the terminal and store in the **termio** structure referenced by *arg*. Only those parameters that can be stored in the **termio** structure are returned. This command is allowed from a process in a background process group; however, the terminal attributes may be subsequently changed by a process in the foreground process group.

TCSETA — Set the parameters associated with the terminal from the **termio** structure referenced by *arg*. Only those terminal parameters that can be stored in the **termio** structure are set with the exception of IEXTEN which is forced to be set. The change is immediate.

TCSETAW — Same as TCSETA except that it first waits for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.

TCSETAF — Same as TCSETA except that it first waits for the output to drain and then flushes the input queue before setting the new parameters.

TCSBRK — Wait for the output to drain. If *arg* (an *int* value) is 0, then send a break (zero bits for 0.25 seconds).

TCXONC — Start/stop control. If *arg* (an *int* value) is 0, suspend output; if 1, restart suspended output; if 2, send a STOP character to the terminal device; if 3, send a START character to the terminal device.

TCFLSH — If *arg* (an *int* value) is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

TIOCGPGRP — Get the process group ID of the foreground process group associated with the terminal and store in the *pid_t* referenced by *arg*. If the terminal referenced by *fildes* is not the controlling terminal of the calling process, the **ioctl**() returns –1 with

errno set to [ENOTTY]. If the lifetime of the current foreground process group has ended, the **ioctl**() returns –1 with *errno* set to [EACCES]. This command is allowed from a process in a background process group; however, the information may be subsequently changed by a process in the foreground process group.

TIOCSPGRP Set the foreground process group ID associated with the terminal from the process group ID stored in the *pid_t* referenced by *arg*. If the terminal referenced by *fildes* is not the controlling terminal of the calling process and is not also still associated with the session of the calling process, the **ioctl**() returns –1 with *errno* set to [ENOTTY]. If the new process group ID value does not match the process group ID of a process in the same session as the calling process, the **ioctl**() returns –1 with *errno* set to [EPERM].

TIOCGWINSZ Get the current **winsize** structure settings. struct **winsize** is defined in **<termio.h>**.

TIOCSWINSZ Set the current **winsize** structure. If any of values differ from their previous values then SIGWINCH is generated.

TIOCGETP Get the parameters associated with the terminal, and store it in the **sgttyb** structure referenced by *arg*. This **ioctl**() is obsolete and **tcgetaddr**() should be used instead.

TIOCSETP Set the parameters associated with the terminal from the **sgttyb** structure referenced by *arg*. The interface delays until output is quiescent, then throws away any unread characters, before changing the modes. This **ioctl**() is obsolete and **tcsetaddr**() should be used instead.

The following POSIX-conforming interfaces are also provided for performing some of the **ioctl**() command functions. Unless otherwise indicated, their return value and *errno* values are identical to the indicated **ioctl**() functions they perform.

```
#include <termios.h>

int tcgetattr (fildes, termios_p)
int fildes;
struct termios *termios_p;
```

The **tcgetattr**() function performs the TCGETP command.

```
#include <termios.h>

int tcsetattr (fildes, optional_actions, termios_p)
int fildes, optional_actions;
struct termios *termios_p;
```

If *optional_actions* is TCSANOW, the function performs the TCSETP command. If *optional_actions* is TCSADRAIN, the function performs the TCSETPW command. If *optional_actions* is TCSAFLUSH, the function performs the TCSETPF command.

```
#include <termios.h>

int tcsendbreak (fildes, duration)
int fildes;
int duration;
```

The **tcsendbreak**() function performs the TCSBRK command with an *arg* of 0.

```
#include <termios.h>

int tcdrain (fildes)
int fildes;
```

The **tcdrain**() function performs the TCSBRK command with an *arg* of 1.

```
#include <termios.h>

int tcflush (fildes, queue_selector)
int fildes;
int queue_selector;
```

If *queue_selector* is TCIFLUSH, the function performs the TCFLSH command with an *arg* of 0. If *queue_selector* is TCOFLUSH, the function performs the TCFLSH command with an *arg* of 1. If *queue_selector* is TCIOFLUSH, the function performs the TCFLSH command with an *arg* of 2.

```
#include <termios.h>

int tcflow (fildes, action)
int fildes;
int action;
```

If *action* is TCOOFF the function performs the TCXONC command with an *arg* of 0. If *action* is TCOON, the function performs the TCXONC command with an *arg* of 1. If *action* is TCIOFF, the function performs the TCXONC command with an *arg* of 2. If *action* is TCION, the function performs the TCXONC command with an *arg* of 3.

```
#include <sys/types.h>

pid_t tcgetpgrp (fildes)
int fildes;
```

The *tcgetpgrp*() function performs the TIOCGPGRP command. The foreground process group ID is returned as the value of the *tcgetpgrp*() function.

```
#include <sys/types.h>

int tcsetpgrp (fildes, pgrp_id)
int fildes;
pid_t pgrp_id;
```

The *tcsetpgrp*() function performs the TIOCSPGRP command. Note that *pgrp_id* contains the value of the new process group ID while *arg* for TIOCSPGRP contains a pointer to the value of the new process group ID.

FILES

/dev/tty*

SEE ALSO

stty(1), fork(2), ioctl(2), setpgrp(2), signal(2)

NAME

tg – SCSI 1/2-inch magnetic tape interface

CONFIGURATION ENTRY

tg on ssm_scsi *ssm_scsi* **unit** *unit* **segsize 32768 nsegs 1**

DESCRIPTION

This device driver supports SCSI target adapters for 1/2-inch magnetic tape drives. All functionality as described in **mtio(7)** is supported, except where noted. Up to eight tape drive units spread across the SSM boards in the system are supported. The SCSI interface for the SSM board supports the target adapter on the SSM board plus seven additional target adapters.

The SCSI 1/2-inch tape is located on SSM SCSI bus *ssm_scsi* and its assigned configuration unit number is *unit*. The **unit** number, if not wild-carded, corresponds to the target adaptor number of the device and also is an index into the *ssm_scsi_config* table in the file */etc/conf/uts/io/ssm/ssm_space.c*. The unit number must be unique with respect to other devices on this SSM SCSI bus. Both the *ssm_scsi* and *unit* may be specified with a "?," meaning they are wild-carded. If *ssm_scsi* is wild-carded, the system will assign the number of the SSM SCSI bus on which the tape is physically located. If *unit* is wild-carded, the system will assign the configuration unit number of the first unused matching entry in the *ssm_scsi_config* table. A tape matches if its target adapter number matches the target adapter number specified in the *ssm_scsi_config* table entry. The *segsize* field is the size of address space reserved for DMA transfers and must be 32768. The *nsegs* field indicates how many regions to reserve for DMA transfers for this device and must be 1.

If a write is performed as the last operation before a **tg** device is closed, two tapemarks will be written to the tape immediately after the last written data. If the tape drive is not to be rewound after closing, the tape head will be positioned between the tape marks.

The standard device names begin with **tg** followed by a number representing the tape unit, that is, **tg0** through **tg7**. The corresponding no-rewind device name has an **n** appended to the standard name, that is, **tg0n** through **tg7n**.

BINARY CONFIGURATION

The *tg_bconf* array in the device driver binary configuration file (*/etc/conf/uts/io/tg/tg_space.c*) must contain an entry for each drive that may be configured; the entries must be in the same order as the "device" entries in the kernel configuration file. As installed, *tg_bconf* has entries for 7 drives. Unused entries are ignored.

Tuning Parameters

The following tuning parameters are defined for each drive in the *tg_bconf* array of the binary configuration file:

(1) Flags for configurable features. The flags available are:

TMC_PRSENSE

When this flag is set, all SCSI request sense buffers will be printed out, along with the command that caused the sense condition. Otherwise, only sense buffers that are in some way unusual will be printed out, such as media and hardware errors.

TMC_RSENSE
When this flag is set, all SCSI request sense buffers associated with SCSI READ operations that caused a sense condition will be printed out.

TMC_WSENSE
When this flag is set all SCSI request sense buffers associated with SCSI write and write filemark operations that caused a sense condition will be printed out.

TMC_SSENSE
When this flag is set all SCSI request sense buffers associated with SCSI space operations which caused a sense condition will be printed out.

FILES

/etc/conf/uts/io/tg/tg_space.c	Raw magtape devices
/dev/rmt/tg*	Raw magtape devices
/etc/conf/uts/io/tg/tg_space.c	Raw magtape devices
/dev/mt/tg*	Block magtape devices
/etc/conf/uts/io/tg/tg_space.c	Binary configuration file

SEE ALSO

mt(1), mtio(7)

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 which supports the Transport Interface. 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 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)

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, then 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 will cause an error on the *stream*. Once the error is detected, subsequent system calls on the *stream* will return 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 will check any existing data destined for the user to ensure that only regular data messages are present. It will ignore any messages on the *stream* that relate to process management, such as messages that generate signals to the user processes associated with the *stream*. If any other messages are present, the I_PUSH will return an error with *errno* set to EPROTO.

write – The module will take 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) will be transparently passed onto the module's downstream neighbor.
- Any zero length data messages will be freed by the module and they will not be passed onto the module's downstream neighbor.
- Any messages with control portions will generate an error, and any further system calls associated with the *stream* will fail with *errno* set to EPROTO.

read – The module will take 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) will be transparently passed onto the module's upstream neighbor.
- The action taken on messages with control portions will be as follows:
  + Messages that represent expedited data will generate an error. All further system calls associated with the *stream* will fail 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 will be freed by the module and they will not be passed onto the module's upstream neighbor.

pop – When the module is popped off the *stream* or the *stream* is closed, the module will take the following action:

– If an orderly release indication has been previously received, then an orderly release request will be 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)

NAME

tm – SCSI tape controller

CONFIGURATION ENTRY

tm on ssm_scsi *ssm_scsi* **unit** *unit* **segsize 131072nsegs 2**

DESCRIPTION

This device driver supports SCSI target adapters for streaming tape drives. It supports up to eight tape drive units spread across the SSM boards in the system. The SCSI interface for the SSM board supports the target adapter on the SSM board plus seven additional target adapters.

The SCSI tape is located on SSM SCSI bus *ssm_scsi* and is configuration unit number *unit*. The **unit** number, if not wild-carded, corresponds to the target adaptor number of the device and also is an index into the *ssm_scsi_config* table in the file */etc/conf/uts/io/ssm/ssm_space.c*. The unit number must be unique with respect to other devices on this SSM SCSI bus. Both the *ssm_scsi* and *unit* may be specified with a "?," meaning they are wild-carded. If *ssm_scsi* is wild-carded, the system will assign the number of the SSM SCSI bus on which the tape is physically located. If *unit* is wild-carded, the system will assign the configuration unit number of the first unused matching entry in the *ssm_scsi_config* table. A tape matches if its target adapter number matches the target adapter number specified in the *ssm_scsi_config* table entry. The *segsize* field is the size of address space reserved for DMA transfers and must be 131072. The *nsegs* field indicates how many regions to reserve for DMA transfers for this device and must be 2.

The standard device names begin with **tm** followed by a number representing the tape unit, that is, **tm0** through **tm7**. The corresponding no-rewind device name has an **n** appended to the standard name, that is, **tm0n** through **tm7n**.

Implications of Streaming

Because the **tm** driver supports streaming tapes, the interface to the **tm** driver is different from that of 'standard' tape drives, in the following ways:

Block interface

The block interface is not supported at all. Attempts to use it will return ENXIO.

Raw interface

To provide reasonable performance and drive life, transfers to and from the tape must be done in large chunks. The **tm** driver assures this by buffering data internally. One consequence of this is that errors on writes often cannot be reported until long after the user passed the data to the driver. Write errors will be reported to the user program on the next write system call after the error occurred, or on the close system call following the error.

The raw interface does not support block sizes like the standard tape does. The **tm** tape device has a 512-byte physical block size. User records are packed into these physical blocks on a write and unpacked from these blocks on a read. When reading, the user gets the number of bytes requested, until end-of-file is encountered. If less than a multiple of 512 bytes was written, the file size will be rounded up to a multiple of 512 bytes. The extra bytes contain zeros.

Reading part-way into a file on the no-rewind device, closing the device, and re-opening it will not work as expected. The tape will always space forward to the beginning of the next file on a close.

Operational Information

Normally, a tape is retensioned when it is first opened for a **read** or **ioctl** call. Likewise, it is erased when it is first opened for a **write** call. This behavior is binary configurable (see Tuning Parameters). Note that if the first operation performed on the tape after insertion (or booting) is '**mt** retension', the tape will be retensioned twice. This automatic retensioning of the tape helps ensure good data transfer, especially on writes to the tape.

The data capacity of a tape will decrease with the number of soft errors on the tape and the number of times the drive must reposition between writes to the tape. These consume tape space. A reasonable expectation for a 600-foot tape is about 60 Mbytes.

The mean time between failure of the tape drive is also affected by the number of repositions while reading or writing the tape. The tape drive will "stream" better when it is accessed under a light system load.

The tape is rewound when a standard device (**tm0** through **tm7**) is closed. This implicit rewind operation is performed asynchronously, that is, the **close(2)** system call may return although the tape is still rewinding. Subsequent attempts to access the drive will wait for the asynchronous operation to complete, but other processing will not be delayed.

IOCTLS

A subset of the ioctl functions are supported at this time. Here is a list of supported and non-supported functions:

MTFSF	Supported.	(**mt fsf** *nn*)
MTWEOF	Supported.	(**mt weof** *nn*)
MTFSR	Not supported.	
MTBSR	Not supported.	
MTSEOD	Supported.	(**mt eod**)
MTREW	Supported.	(**mt rewind**)
MTERASE	Supported.	(**mt erase**)
MTOFFL	Not supported.	
MTBSF	Not supported.	
MTRET	Supported.	(**mt retension**)

The MTERASE ioctl erases the entire tape and then rewinds it. Reading status (**mt** status) is not supported.

ioctls will not always behave as expected if they are intermixed with reads and writes in a program.

BINARY CONFIGURATION

The *tm_bconf* array in the device driver binary configuration file (*/etc/conf/uts/io/tm/tm_space.c*) must contain an entry for each drive that may be configured; the entries must be in the same order as the "device" entries in the kernel configuration file. As installed, *tm_bconf* has entries for 4 drives. Superfluous entries are ignored.

Tuning Parameters

The following tuning parameters are defined for each drive in the *tm_bconf* array of the binary configuration file:

(1) The buffer size (in K bytes) used to buffer data for I/O transfers. Two buffers, each of this size, are allocated from the kernel's free page pool when the device is opened. These buffers are used to improve tape streaming performance. The buffer space is released when the device is closed.

(2) Flags for configurable features. The flags available are:

TMC_OPENFAIL
: When this flag is set, open calls to this device can fail for lack of physical memory for the buffers. When this flag is clear, the buffer memory will always be allocated, but the process may block during this allocation process while other processes are swapped out to free up memory.

TMC_AUTORET
: When this flag is set, the device will retension automatically on the first access after the tape has been changed. When clear, it will not retension automatically.

TMC_PRSENSE
: When this flag is set, all SCSI request sense buffers will be printed out, along with the command that caused the sense condition. Otherwise, only sense buffers that are in some way unusual will be printed out, such as media and hardware errors.

TMC_RSENSE
: When this flag is set, all SCSI request sense buffers associated with SCSI READ operations that caused a sense condition will be printed out.

TMC_WSENSE
: When this flag is set, all SCSI request sense buffers associated with SCSI write and write filemark operations that caused a sense condition will be printed out.

TMC_SSENSE
: When this flag is set, all SCSI request sense buffers associated with SCSI space operations which caused a sense condition will be printed out.

(3) bits: This should always be a 1 for proper operation of fixed block devices.

FILES

/etc/conf/uts/io/tm/tm_space.c	Raw files
/dev/rmt/tm*	Raw files
/etc/conf/uts/io/tm/tm_space.c	Binary configuration file

SEE ALSO

mt(1), mtio(7), ts(7)

WARNINGS

Determining when the tape can safely be removed from the drive is difficult after closing a standard device (**tm0** through **tm7**). The implicit tape rewind must complete prior to media removal. After closing, subsequent attempts to access the drive, such as with an **mt noret** command, will not complete until the rewind has finished.

BUGS

The double file mark at the end of the tape cannot be supported.

NAME

ts – SCSI tape controller

CONFIGURATION ENTRY

ts on sec_scsi *sec_scsi* **unit** *unit*

DESCRIPTION

This device driver supports SCSI target adapters for streaming tape drives. It supports up to four tapes per target adapter and a maximum of seven target adapters per SCED board, not to exceed eight tapes in the entire system.

The SCSI tape is located on SCED SCSI bus *sec_scsi* and is configuration unit number *unit*. The configuration unit number is an index into the *sec_scsi_config* table in the file */etc/conf/uts/io/sec/sec_space.c*, which specifies the target adapter number. The unit number must be unique with respect to other devices on this SCED SCSI bus. Both the *sec_scsi* and *unit* may be specified with a "?," meaning they are wild-carded. If *sec_scsi* is wild-carded, the system will assign the number of the SCED SCSI bus on which the tape is physically located. If *unit* is wild-carded, the system will assign the configuration unit number of the first unused matching entry in the *sec_scsi_config* table. A tape matches if its target adapter number matches the target adapter number specified in the *sec_scsi_config* table entry.

The standard device names begin with **ts** followed by a number representing the tape unit, that is, **ts0** through **ts7**. The corresponding no-rewind device name has an **n** appended to the standard name, that is, **ts0n** through **ts7n**.

Implications of Streaming

Because the **ts** driver supports streaming tapes, the interface to the **ts** driver is different from that of 'standard' tape drives, in these ways:

Block interface

The block interface is not supported at all. Attempts to use it will return ENXIO.

Raw interface

To provide reasonable performance and drive life, transfers to and from the tape must be done in large chunks. The **ts** driver assures this by buffering data internally. One consequence of this is that errors on writes often cannot be reported until sometime long after the user passed that data to the driver. Write errors will be reported to the user program on the next write system call after the error occurred, or on the close system call following the error.

The raw interface does not support block sizes in the same fashion that standard tape does. The **ts** tape device has a 512-byte physical block size. User records are packed into these physical blocks on write and unpacked from these blocks on **read**. So, when reading, the user gets as many bytes as he asks for, until end-of-file is encountered. If less than a multiple of 512 bytes was written, the file size will be rounded up to a multiple of 512 bytes. The extra bytes contain zeros.

Reading part-way into a file on the no-rewind device, closing the device, and re-opening it will not work as expected. The tape will always space forward to the beginning of the next file on close.

Operational Information

A tape is always retensioned when it is first opened for **read** or **ioctl**. It is erased when it is first opened for write. Note that if the first operation performed on the tape after insertion (or booting) is **mt retension**, the tape will be retensioned twice. This automatic retensioning of the tape is to help ensure good data transfer, especially on writes to the tape.

The data capacity of a tape will decrease with the number of soft errors on the tape and the number of times the drive must reposition between writes to the tape. These consume tape space. A reasonable expectation for a 450-foot tape is about 44 Mbytes.

The mean time between failure of the tape drive is also affected by the number of repositions while reading or writing the tape. The tape drive will 'stream' better when it is accessed under light system load.

IOCTLS

A subset of the ioctl functions are supported at this time. Here is a list of supported and non-supported functions:

MTFSF	Supported	**(mt fsf** *nn***)**
MTWEOF	Supported	**(mt weof** *nn***)**
MTFSR	Not supported	
MTBSR	Not supported	
MTSEOD	Supported	**(mt eod)**
MTREW	Supported	**(mt rewind)**
MTERASE	Supported	**(mt erase)**
MTOFFL	Not supported	
MTBSF	Not supported	
MTRET	Supported	**(mt retension)**

The MTERASE **ioctl** erases the entire tape and then rewinds it. Reading status (**mt status**) is not supported.

Ioctls will not always behave as expected if they are intermixed with reads and writes in a program.

BINARY CONFIGURATION

The **tsbconf** array in the device driver binary configuration file (*/etc/conf/uts/io/ts/ts_space.c*) must contain an entry for each drive that may be configured; the entries must be in the same order as the 'device' entries in the kernel configuration file. As installed, **tsbconf** has entries for 4 drives. Superfluous entries are ignored.

Tuning Parameters

The following tuning parameters are defined for each drive in the **tsbconf** array of the binary configuration file:

(1) The buffer size (in K bytes) used to buffer data for I/O transfers. Two buffers, each of this size, are allocated from the kernel's free page pool when the device is opened. These buffers are used to improve tape streaming performance. The buffer space is released when the device is closed.

(2) Flags for configurable features. Flags available are:

TSC_OPENFAIL
When this flag is set, open calls to this device can fail for lack of physical memory for the buffers. When this flag is clear, the buffer memory will always be allocated, but the process may block during this allocation process while other processes are swapped out to free up memory.

TSC_AUTORET
When this flag is set, the device will retension automatically on the first access after the tape has been changed. When clear, it will not retension automatically.

TSC_PRSENSE
When this flag is set, all SCSI request sense buffers will be printed out, along with the command which caused the sense condition. Otherwise, only sense buffers which are in some way unusual will be printed out.

(3) bits: For the ADSI and Emulex target adapters, this should always be a 1 for proper operation of the device.

FILES

/etc/conf/uts/io/ts/ts_space.c	Raw file
/dev/rmt/ts*	Raw file
/etc/conf/uts/io/ts/ts_space.c	Binary configuration file

SEE ALSO

mt(1), mtio(7), sd(7)

BUGS

The double file mark at the end of the tape cannot be supported.

NAME

tty – controlling terminal (gentty) interface

CONFIGURATION ENTRY

pseudodevice gentty

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.

The controlling terminal is established only by a controlling process. Such a process must be a session leader and must not already have established a controlling terminal. The terminal must not already be someone else controlling terminal, in which case an open without O_NOCTTY set will establish it.

When the controlling terminal is relinquished, subsequent reads and writes will return ENXIO and subsequent opens will return EIO.

FILES

/dev/tty
/dev/tty*

SEE ALSO

console(7), dev_to_ttyname(3C)

NAME

usclk - 32-bit microsecond counter

CONFIGURATION ENTRY

pseudodevice usclk

DESCRIPTION

This device provides access to a 32-bit counter which increments every microsecond. This counter can be used to do fine-grain timing studies on program execution. This device does not support **read()** and **write()** system call access. To access the counter, this device must be mapped into a process's memory (see **mmap**(2seq)). Only one page of usclk memory can be mapped into a process (see **getpagesize**(2seq)). Requests to **mmap** more will fail. The device can only be mmaped READ ONLY. Generally, this device should be accessed only through the **usclk_init**(3seq) and **getusclk**(3seq) library functions or the **GETUSCLK** macro defined in */usr/include/usclkc.h*.

Symmetry systems implement this counter as a set of 32-bit registers on the processor boards. Each processor has its own counter register. But, the initialization and incrementing of these counter registers is synchronized across all processor boards. The resulting effect is as though there were only one counter register. However, accessing this counter involves no accesses off the processor board. So, by mapping the device into the process's address space, the overhead of accessing the counter is essentially zero.

DIAGNOSTICS

EINVAL An attempt was made to map with write access.

ENOSPC An mmap request tried to map more than one page of the usclk device. Refer to **getpagesize**(2seq).

ENXIO An open request is denied because */dev/usclk* has a minor device number greater than one.

NAME

vtoc – VTOC partitioning driver

CONFIGURATION ENTRY

pseudodevice vtoc

DESCRIPTION

This device driver provides the VTOC style of disk partitioning, which means that access is provided to the underlying disk space via partitions which are described by the disk itself. This is a "virtual" device driver, in that it requires an actual disk device (such as a **zd**, **sd**, or **wd** device) which contains a valid VTOC in the correct location to build upon.

Disk device drivers do not provide any partitions, but present the entire disk surface for use. To make this so-called "unpartitioned" disk more useful, the disk surface may be partitioned into up to 255 non-overlapping sections. Each of these disk sections are accessed via a separate minor number.

Special files with minor device numbers 0 through 254 refer to various portions of drive 0; minor devices 256 through 510 refer to drive 1, etc. The special files with minor number 255 refers to drive 0 in the unpartitioned state, and is not accessible if a partitioning driver is in place on this disk. The standard device names begin with the disk controller name (e.g., **zd** or **wd**), followed by the drive number, followed by the letter **s**, and then a number from 0-254 — e.g., **zd0s0**. By convention, these special files are found in the directory */dev/dsk*.

The block devices access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a *raw* interface that provides for direct transmission between the disk and the user's read or write buffer. Raw transfers are limited to a maximum raw transfer size. The maximum raw transfer size is specific to each driver, but is usually 256 512-byte sectors. In raw I/O, counts must be a multiple of 512 bytes, and must begin on 16-byte aligned memory addresses. Likewise, **lseek** calls should specify a multiple of 512 bytes. A single read or write call usually results in exactly one I/O operation, which transfers data directly to the users address space. Therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw special files are conventionally the same as the block special files, and are found in the directory **/dev/rdsk** — e.g., */dev/rdsk/wd0s0*. In addition to these regular device interfaces, there is a "diagnostic" interface to the disk once the VTOC driver is present. This interface will allow VTOC ioctls to be issued and also provides a read-only interface to the entire unpartitioned disk address space. The special file for the diagnostic interface has a major and minor device number equal to the unpartitioned device with 32768 added to the major number. By convention, this special file is in the directory */dev/diag/rdsk* — e.g., */dev/diag/rdsk/sd0*.

Note that when the **vtoc** driver is in place on a disk, the unpartitioned interface to the disk becomes "unavailable," which means that all attempts to open the unpartitioned disk will fail with ENXIO.

Partition Data

The partition data resides in an area of the disk called the "Volume Table of Contents" (VTOC) which is at most V_SIZE sectors long, and resides at sector

V_VTOCSEC, offset by any reserved cylinders a disk may have.

The structure of the VTOC is described in */usr/include/sys/vtoc.h* . Some of the fields include:

v_sanity
This is a magic cookie which identifies this section of the disk as a valid VTOC. It is normally set to VTOC_SANE.

v_version
The version of this particular VTOC.

v_cksum
A checksum, which allows the contents of the VTOC to be verified.

v_nparts
The number of valid partitions in this VTOC.

v_secsize
The size in bytes of sectors on this disk.

v_ntracks
The number of tracks per cylinder.

v_nsectors
The number of sectors per track.

v_ncylinders
The number of cylinders on this disk. The parameters *v_ntracks*, *v_nsectors*, and *v_ncylinders* are used to make filesystems. Since it is not always possible to know the exact geometry of a disk or a disk may not have a fixed geometry, these values should be chosen to reflect the total size of the disk when multiplied together.

v_rpm The number of disk revolutions-per-minute.

v_disktype
This is a character array which gives the name of the disk type. It is at most VTYPCHR characters long.

v_part Per-partition data.

Each partition has data associated with it. That information includes:

p_start The sector number where this partition starts.

p_size The size of this partition in sectors.

p_type The type of this partition.

p_bsize
The size in bytes of filesystem blocks to be made for this partition.

p_fsize The filesystem fragment size in bytes for this partition.

The parameters *p_bsize* and *p_fsize* will only be needed if a filesystem is to be made on this partition. If no filesystem will be made on this partition, then these values are not significant.

Partition Types

Each partition can have associated with it a partition type, which is recorded in the per-partition section of the VTOC. The partition types are:

V_NOPART
This type indicates that this slot in the partition table is empty. No device will be made available which corresponds to this minor device number.

V_RAW This is a regular partition. Partitions of this type may have filesystems constructed on them, or may be used as raw disk partitions.

V_BOOT
System bootstrap area. These partitions are like partitions of type V_RAW, but they have added meaning in that commands which write system bootstraps to disk will by default look for the first partition of type V_BOOT to write the bootstrap to.

V_RESERVED
This designates an area of the disk that may not be accessed under normal circumstances, since it maps an area of the disk reserved for some specific purpose. For example, some disk controllers require that areas of the disk be reserved for bad block lists or configuration information. Most disk types also reserve space for diagnostic routines to read and write the disk without destroying data. The VTOC itself is mapped as reserved space. Programs that write VTOCs directly should disallow any attempt to change the definition of partitions of this type, or serious problems may result.

V_FW System controller firmware area. These partitions are like partitions of type V_RAW but they have added meaning in that commands which write system bootstraps to disk will by default look for the first partition of type V_FW to write the firmware to. The firmware is only written if the system has a system controller board with loadable firmware.

IOCTLS

This driver supports the following commands to *ioctl()*:

V_BUILD
This ioctl is used to build a VTOC device from an unpartitioned device. It should be passed to the device which is normally called **/dev/vtoc** and have, as the third argument to the system call, an open file descriptor for a disk which does not have a partitioning driver in place. A possible coding example using this ioctl would be:

```
fd = open("/dev/vtoc", O_RDWR);
fd1 = open("/dev/rdsk/zd1", O_RDWR);
ioctl(fd, V_BUILD, fd1);
```

If the ioctl is successful, the file descriptor passed as the third argument should be treated as if it has already been closed and a subsequent **close()** operation is not needed on this descriptor. This is necessary, since a successful V_BUILD operation will make the unpartitioned device inaccessible. (i.e., */dev/rdsk/zd1* can no longer be opened if the V_BUILD operation above is successful).

V_DESTROY
The VTOC device in place on a disk is instructed to destroy all of the

devices associated with partitions on this disk. The unpartitioned device is made available for use again. This ioctl will fail with EBUSY if any of the partitions which make up this disk are open, being used as a swap device, or has a mounted filesystem. The semantics of this ioctl are identical to the V_BUILD ioctl, with the exception that the third argument to the ioctl call is the file descriptor from opening any valid partition on this disk or from opening the diagnostic device for this disk.

V_READ

This ioctl will cause the VTOC to be read from disk and returned to the caller. This is the only way to read the VTOC, since read/write access to the VTOC area is disallowed when a VTOC driver is present. The third argument to the ioctl call should be a pointer to an area of memory large enough to store the entire *struct vtoc*. For portability to future versions of the kernel, a block of size **V_SIZE** should be allocated for *struct vtoc*.

V_WRITE

This ioctl will cause the VTOC to be written to disk. This is the only way to write the VTOC, since read/write access to the VTOC area is disallowed when a VTOC driver is present. The third argument to the ioctl call should be a pointer to a filled-out *struct vtoc* which will be sanity-checked before being written to disk. If the VTOC fails a sanity-check, then EINVAL is returned. This may include overlapping partitions, or an attempt to put a partition over the VTOC area of the disk which is not of type V_RESERVED. If a partition is changed in the new VTOC relative to the old VTOC (i.e., any of the per-partition data is changed), that partition must not be mounted or being used as a swap device, and the raw device must not be open by any process.

V_WRITEPROTECT

This ioctl acts as a software write-protect switch. The first argument to the ioctl call should be an open file-descriptor to a raw partition. When V_WRITEPROTECT is run, all subsequent attempts to open for writing the character-special or block-special device for this partition will result in EBUSY being returned in errno. Note that the *mount*(2) system call performs the equivalent of a V_WRITEPROTECT ioctl to the mounted device.

V_WRITEENABLE

This ioctl reverses the effect of the V_WRITEPROTECT ioctl, allowing the partition's character and block-special devices to be opened for writing.

DEVTAB ENTRY

In order for the **vtoc** driver to get established on disks automatically at system boot time, the file */dev/devtab* must have lines added which direct the binding of the vtoc driver to the disk. Lines for **vtoc** in */etc/devtab* have the following form:

vtoc *special*

where *special* is a character special file for an unpartitioned physical disk device. If the **vtoc** driver becomes successfully established on the disk, *special* will become unavailable for use and the partitioned devices themselves become usable.

For example, consider the following lines from */etc/devtab*:

```
vtoc   /dev/rdsk/zd1
vtoc   /dev/rdsk/sd0
vtoc   /dev/rdsk/sd1
```

These lines will automatically build a vtoc driver on **zd1**, **sd0**, and **sd1**. Note that the root disk should not be listed, since a **vtoc** driver is established on the root disk automatically. In addition, if the primary swap device is on a different disk than the root, then a **vtoc** driver gets established on this disk automatically as well.

FILES

/dev/vtoc
/dev/dsk/*
/dev/rdsk/*
/dev/diag/rdsk/*
/etc/devtab

SEE ALSO

cmpt(7), devbuild(1M), devtab(4)

NAME

wd – SSM SCSI disk controller

CONFIGURATION ENTRY

wd on ssm_scsi *ssm_scsi* **unit** *unit* **segsize 131072nsegs 2**

DESCRIPTION

This device driver controls SCSI disk with embedded targets ("embedded SCSI" disks). It supports up to seven embedded SCSI disks per SSM board. The SCSI disk is located on SSM SCSI bus *ssm_scsi* and is configuration unit number *unit*. The **unit** number, if not wild-carded, corresponds to the target adapter number of the device and also is an index into the *ssm_scsi_config* table in the file */etc/conf/uts/io/ssm/ssm_space.c*. The unit number must be unique with respect to other devices on this SSM SCSI bus. Both the *ssm_scsi* and *unit* may be specified with a "?," meaning they are wild-carded. If *ssm_scsi* is wild-carded, the system will assign the number of the SSM SCSI bus on which the disk is physically located. If *unit* is wild-carded, the system will assign the configuration unit number of the first unused matching entry in the *ssm_scsi_config* table. A disk matches if its target adapter number matches the target adapter number specified in the *ssm_scsi_config* table entry. The *segsize* field is the size of address space reserved for DMA transfers and must be 131072. The *nsegs* field indicates how many regions to reserve for DMA transfers for this device and must be 2.

The special file with minor device number 255 refers to drive **wd0** (the first drive listed in the kernel configuration file), minor device number 511 refers to **wd1**, and so on. Standard device names are of the form **wd***n*, where *n* is the drive number.

Block devices access the disk via the system's normal buffering mechanism, and may be read and written without regard to physical disk records. These special files are usually found in **/dev/dsk** — e.g., **/dev/dsk/wd0**. There is also a *raw* interface that provides for direct transmission between the disk and the user's read or write buffer. Raw transfers are limited to a maximum of 256 512-byte sectors. In raw I/O, counts must be a multiple of 512 bytes, and must begin on 16-byte aligned memory addresses. Likewise, **lseek** calls should specify a multiple of 512 bytes. A single read or write call usually results in exactly one I/O operation, which transfers data directly to the users address space. Therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw special files are conventionally the same as the block special files, and are found in the directory **/dev/rdsk** — e.g., **/dev/rdsk/wd0**.

NOTE: The wd driver provides access to the entire disk, including reserved sectors. To make the device usable, a partitioning driver (like **vtoc**(7) or **cmpt**(7)) must be built on the unpartitioned disk.

BINARY CONFIGURATION

Before attempting to add a new disk or target adapter, read the disk and target adapter manuals thoroughly. The main information needed is how to set the target adapter number and drive-select cabling.

Table of disk units.
The **wd_bconf** array in the device driver binary configuration file (*/etc/conf/wd_space.c*) must contain an entry for each drive that may be

configured; the entries must be in the same order as the *device* entries in the kernel configuration file.

Tuning Parameters

Tuning parameters are defined for each drive in the **wd_bconf** array of the binary configuration file. Each element specifies the maximum number **IATs** (indirect access table entries used for scatter/gather I/O) for the corresponding drive unit.

An indirect access table is used to map a contiguous I/O buffer in virtual memory to the corresponding pages scattered throughout physical memory. Each table entry (IAT) can map up to 4096 bytes. The number of IATs determines the maximum transfer size for each read/write request.

Since each drive may simultaneously map a maximum of two 128-Kbyte transfers, the maximum number of IATs every used is 64.

FILES

/dev/dsk/wd?s?	Block files
/dev/rdsk/wd?s?	Raw files
/usr/conf.new/uts/io/wd/wd_space.c	Binary configuration file

SEE ALSO

devbuild(1M), **mkvtoc**(1M), **newfs**(1M), **prtvtoc**(1M), **vtoc**(7), **cmpt**(7), **CCSformat**(8)

DIAGNOSTICS

Embedded controller internal recovery and error reporting

See the ANSI X3T9.2 (SCSI) specification.

Firmware and SCSI bus level errors (reported as HARD errors)

Command Block Busy
Bad Command Block
No Target
SCSI Bus Error

SCSI target adapter errors and actions taken

Recovered Error
    SOFT error message, for information only
Unit Attention
    Totally ignored
Medium Error
Hardware Error
Illegal Request
Data Protect
Blank Check
Vendor Unique
Copy Aborted
Aborted Command
Equal
Volume Overflow
Miscompare
Unrecognized Completion Code
    Retried four times with soft error messages; fifth error

produces Hard Error message

BUGS

In raw I/O, **read**(2),**write**(2), and **lseek**(2) should always deal in 512-byte multiples.

A program to analyze the logged error information (even in its present reduced form) is needed.

The kernel 'milliseconds per seek' statistic is not kept for this device; thus, **iostat**(1) always reports zero for this statistic.

When a request to the disk accesses past the end of media (for example, a non-existent block number), the device driver will report soft errors followed by a hard error in the same way that a bad block is treated. There is no special handling of the volume overflow sense error code for disks.

NAME

wp – SSM parallel line printer interface

DESCRIPTION

The **wp** device driver supports the SSM controller for line printers using the Centronics or Dataproducts parallel interfaces. It supports one line printer per controller.

The unit number of the printer is specified by the minor device number. The standard device names consist of "wp" followed by the line printer unit number (e.g., wp0, wp1). **wp** devices are exclusive-access devices; that is, only one process may open a device at a time.

BINARY CONFIGURATION

The printer can be configured for one of three different modes which are described in **lpmod**(7). The different modes are implemented by **lpmod,** a STREAMS module pushed onto the driver automatically when the device is opened. If the system is configured to not have this module pushed, the printer will always be in *raw* mode regardless of what is in the *lp_config* table. The table which causes this module to be pushed is in */etc/conf/uts/kernel/i386_space/autop_space.c*.

Raw mode provides an eight-bit data path to the printer. Tabs, formfeeds, newlines and other special characters are not expanded, and no line or page adjustments are made.

To see a description of the other modes, refer to the man page for **lpmod.**

The interface type is defined as follows:

Value	Definition
0	Centronics interface
1	Data Products interface

The configuration options are specified in a table in the binary configuration file, */etc/conf/uts/io/wp/wp_space.c*. The first column specifies the line length. The second column specifies the mode. The third column specifies the printer interface type. Note that both the mode and the line length field are ignored by the driver and are only there for the benefit of modules which may be pushed onto the driver. The printer characteristics are indexed by the minor number of the device. If the minor number is greater than the number of entries in the array, the configuration options for the last entry in the table are used.

For example, if you had the following hardware configuration:

```
printer 0     80 columns      uppercase      Centronics
printer 1     132 columns     raw            Data Products
printer 2     132 columns     normal         Data Products
printer 3     80 columns      normal         Centronics
printer 4     132 columns     normal         Centronics
printer 5     132 columns     normal         Centronics
```

you could use the following configuration table:

```
struct wp_printer wpconfig[] = {

/*      cols,       special_map, interface      */
{       80,         PRCAPS,      0          } , /* 0 */
{       132,        PRRAW,       1          } , /* 1 */
{       132,        PRSPECIAL,   1          } , /* 2 */
{       80,         PRSPECIAL,   0          } , /* 3 */
{       132,        PRSPECIAL,   0          } , /* 4 */
};
```

Note that the sixth printer (number 5) is not specified in the configuration but will have the same characteristics as printer number 4.

The default configuration is as follows:

```
struct wp_printer wpconfig[] = {

/*      cols,       special_map, interface      */
{       132,        PRSPECIAL,   0          } , /* 0 */
};
```

FILES

/dev/printer/wp*
/etc/conf/uts/io/wp/wp_space.c

SEE ALSO

lp(1), pr(1), lpadmin(1M), lpstat(1), terminfo(4), lpmod(7)

NAME

xs – Xylogics 781 terminal multiplexor driver

CONFIGURATION ENTRY

xs on vme *vme* **unit** *unit* **segsize 1024 nsegs 16**

DESCRIPTION

The Xylogics 781 board provides 16 serial I/O lines. Each line is able to support a modem or a terminal. The controller board is located on VME bus *vme* and is configuration unit number *unit*. The configuration unit number is an index into the *xs_vme_config* table in the file */etc/conf/uts/io/xs/xs_space.c*. The unit number must be unique with respect to other xs devices on this VME bus. Both the *vme* and *unit* may be specified with a "?," meaning they are wild-carded. If *vme* is wild-carded, the system will assign the number of the VME bus in which the board is physically located. If *unit* is wild-carded, the system will assign the configuration unit number of the first unused matching entry in the *xs_vme_config* table. A board matches if it is at the address specified in the *xs_vme_config* table entry. The *segsize* field is the size of address space reserved for DMA transfers and must be 1024. The *nsegs* field indicates how many regions to reserve for DMA transfers for this device and must be 16.

Each line attached to the 781 terminal multiplexor behaves as described in **streamtty**(7). Input and output for each line may independently be set to run at any of the following speeds: **B50, B75, B110, B150, B300, B600, B1200, B1800, B2400, B4800, B9600,** and **B19200**.

RS232 output line DTR is enabled when the line is opened and disabled when the line is closed if the HUPCL flag is set. The RTS output line is used to control the flow of data coming into the serial port. The CTS input line may be used to control the flow of data coming out of the serial port. The CTS input line floats to logic true on the 781 hardware. Break detection and generation are also supported.

The initial baud rate and line state for all lines are defined by the *xscflags* variable in the binary configuration file, *xs_space.c*. These values are usually overridden (e.g., by **getty**(1a)) when a line is opened.

Streams Driver Characteristics

The **xs** driver is a STREAMS-based tty driver as described in **streamtty**(7). Functionality which is unique to this driver is described below:

When a set-parameters ioctl message is received with a baud rate of B0, this is considered to be a hangup message. DTR is dropped immediately.

If a set-parameters ioctl message is received with an illegal baud rate, the message will be rejected and the line state will not be changed.

The Xylogics 781 hardware is not allowed to perform any flow control. The ld0 module and the driver execute all flow control processing. If the driver receives an M_PCCTL message with a sub-type of MC_STOPI or MC_STARTI, only the first data byte in the message is sent to the peripheral device.

A logic-high-to-logic-low transition on the DCD input is interpreted as a hangup signal.

If the driver receives an M_IOCTL/M_PCIOCTL message with a sub-type of TCGETA/TCGETP and the message does not arrive with a pre-allocated data

buffer, the driver will allocate a streams buffer and attach it to the message.

Refer to your hardware documentation for a description of standard wiring for Sequent terminals and modems.

WARNINGS

The SMC 78C808 Octart chip on the 781 hardware does not detect breaks in an industry-standard way. Therefore, the 781 firmware is required to do some special processing of break interrupts from the Octart in order to support reasonable break detect functionality. Because of this, any terminal connected to a 781 serial port must be set up to for a break duration of no more than 250ms. If a longer break duration is sent to a 781 port, multiple break-detect messages will be sent from the driver to the user process, resulting in strange behavior.

FILES

/dev/tty<subdir>/<basename>
/etc/conf/uts/io/xs/xs_space.c

SEE ALSO

tty(7), streamtty(7)

NAME

xt – Xylogics 472 MULTIBUS magnetic tape interface

CONFIGURATION ENTRY

xt on multibus *multibus* **unit** *unit* **segsize 1024 nsegs 17**

DESCRIPTION

The Xylogics 472 and an industry-standard Cipher formatted interface transport combine to provide a standard tape drive interface as described in **mtio**(7). The controller board is located on MULTIBUS *multibus* and is configuration unit number *unit*. The configuration unit number is an index into the *xt_multibus_config* table in the file */etc/conf/uts/io/xt/xt_space.c*. The unit number must be unique with respect to other xt devices on this MULTIBUS. Both the *multibus* and *unit* may be specified with a "?," meaning they are wild-carded. If *multibus* is wild-carded, the system will assign the number of the MULTIBUS in which the board is physically located. If *unit* is wild-carded, the system will assign the configuration unit number of the first unused matching entry in the *xt_multibus_config* table. A board matches if it is at the address specified in the *xt_multibus_config* table entry. The interrupt vector cannot be checked, so care must be taken to ensure that the interrupt vector is jumpered to match the *xt_multibus_config* table entry. The *segsize* field is the size of address space reserved for DMA transfers and must be 1024. The *nsegs* field indicates how many regions to reserve for DMA transfers for this device and must be 17.

The bits in the minor device number are as follows: The low-order two bits select the logical unit. As many as four transports on any combination of controllers and formatters are supported. The next bit, 004, specifies that the tape is not rewound when the associated special file is closed.

The tape controller must be strapped up for 24-bit addressing.

BINARY CONFIGURATION

When building a kernel with MULTIBUS magtape, be sure that the binary configuration information in */etc/conf/uts/io/xt/xt_space.c* is up to date. The constants and variables defined in *xt_space.c* include:

XTCTLR — The maximum number of **xt** controllers supported by the driver.

xtunits — This array (indexed by logical unit from the minor device number) describes the controller and transport assignments that will be supported by the system. Wildcarding may be used to force the driver to probe each controller for the specified tape drives. The example below describes two possible controllers, with two tape drives on each controller.

```
structxt_unit      xtunits[] = {
   /*  ctlr        drive */
   { ANY,    ANY }, /* xt0: any controller, any drive */
   { ANY,      1 }, /* xt1: any controller, drive 1 */
   { ANY,      0 }, /* xt2: any controller, drive 0 */
   { ANY,      1 }, /* xt3: any controller, drive 1 */
};
```

WARNINGS

Standalone MULTIBUS magtape offers a subset of the facilities offered by UNIX MULTIBUS magtape. In particular, the standalone tape driver does not do density selection on open (the density must be set manually from the tape front panel), and does not do retries on I/O errors.

FILES

/dev/mt/xt*	Block magtape devices
/etc/conf/uts/io/xt/xt_space.c	Block magtape devices
/dev/rmt/xt*	Raw magtape devices
/etc/conf/uts/io/xt/xt_space.c	Binary configuration file

SEE ALSO

mt(1), tar(1), mtio(7)

DIAGNOSTICS

xt%d: drive %d ctlr %d conflict. A drive was not wildcarded in the binary configuration, but the specified drive number was already in use (allocated to some previously probed tape device).

xt%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

xt%d: not online. An attempt was made to access the tape while it was off-line; this message is written on the terminal of the user who tried to access the tape.

xt%d: can't switch density in mid-tape. An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

xt%d: hard error bn%d er=0x%x. A tape error occurred at block *bn*. The error code is as described in the Xylogics manual. Any error is fatal on non-raw tape; when possible the driver or controller will have retried the operation which failed several times before reporting the error.

xt%d: recovered error bn%d. The controller was able to recover an error while reading or writing.

BUGS

If any non-data error is encountered on non-raw tape, it refuses to do anything more until closed.

Due to a hardware limitation, the xt tape driver cannot tell the difference between a missing tape drive and a tape drive with no tape loaded. As a consequence, tape probing pretends it found all configured tape drives on any controller present in the system. Be careful with wildcarding in the binary configuration.

If a drive supports more than one tape density, the user must manually select the desired tape density while loading the tape.

NAME

zd – dual-channel disk controller

CONFIGURATION ENTRY

zdc *ctrl* **on sb8000 unit ?**
smde *busno* **on zdc** *ctrl* **unit ?**
zd on smde *busno* **unit** *unit*

DESCRIPTION

This device driver controls SMD drives on the Sequent Dual-Channel Disk Controller (DCC). Each DCC can control up to eight SMD disk drives split across two DMA channels. All of the drives on the same channel must be identical in type and format.

The controller is specified by *crtl*. The controllers must be numbered sequentially starting at 0. The unit in the **zdc** configuration statement must be wildcarded. In the **smde** configuration statement the *busno*, SMD bus number, must be numbered sequentially starting at 0 and the unit must be wildcarded. The configuration unit number is an index into the *zd_smde_config* table in the file */etc/conf/uts/io/zd/zd_space.c*. The unit number must be unique with respect to other **zd** devices on this SMD bus. The *unit* may be specified with a "?," meaning they are wild-carded. If *unit* is wild-carded, the system will assign the configuration unit number of the first unused matching entry in the *zd_smde_config* table. A drive matches if its drive number and drive type match the entry in the *zd_smde_config* table entry. In the *zd_smde_config* table either the drive number or the drive type will be be wildcarded, but not both.

The special file with minor device number 255 refers to drive 0; minor device 511 refers to drive 1, and so forth. The standard device names begin with **zd** followed by the drive number — e.g., **zd0**. By convention, these special files are found in the directory */dev/dsk*.

The block devices access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a *raw* interface that provides for direct transmission between the disk and the user's read or write buffer. Raw transfers are limited to a maximum of 256 512-byte sectors. In raw I/O, counts must be a multiple of 512 bytes, and must begin on 16-byte aligned memory addresses. Likewise, **lseek** calls should specify a multiple of 512 bytes. A single read or write call usually results in exactly one I/O operation, which transfers data directly to the users address space. Therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw special files are conventially the same as the block special files, and are found in the directory **/dev/rdsk** — e.g., **/dev/rdsk/zd0**.

NOTE: The zd driver provides access to the entire disk, including bad block lists, configuration data, and diagnostic test tracks. To make the device usable, a partitioning driver (like **vtoc**(7) or **cmpt**(7)) must be built on the unpartitioned disk. Use of a zd device without a partitioning driver in place will result in system errors if the first sectors of the device are accessed.

A drive may be spun up *after* the operating system is booted if the configuration for the drive is *anchored* (controller and drive numbers explicitly specified in the kernel configuration file) and there are no *wildcarded* drives configured

before the *anchored* drive. This is possible since the state of the disk drive and configuration is reassessed at each first open of the device.

BINARY CONFIGURATION

When building a kernel with DCC disks configured, be sure that the binary configuration information in */etc/conf/uts/io/zd/zd_space.c* is up to date. The constants and variables defined in *conf_zd.c* include:

zdc_iovpercb
: This is the number of iovecs allocated per DCC command block. See */usr/include/sys/zdc.h*. If the value is set to zero, the driver will allocate enough iovecs to handle the system's notion of the maximum raw I/O transfer size (i.e., max_RAW_IO). Each iovec contains the physical address of a page in memory. The driver minphys routine (zdminphys()) uses this variable to set the maximum raw I/O transfer size.

zdretry
: The maximum number of times to retry errors.

Disk Support

The driver supports the Fujitsu M2333K "Swallow," the Fujitsu M2344K "Swallow 4," the Fujitsu M2382K "Swallow 5," and the Fujitsu M2351A "Eagle" disk drives. Other drives may be added to the configuration tables. The first cylinder (cylinder 0) is reserved to contain the bad block lists and disk description data. The last 2 cylinders are reserved for diagnostics.

FILES

/dev/dsk/zd[0-31]	Block files
/dev/rdsk/zd[0-31]	Raw files
/etc/conf/uts/io/zd/zd_space.c	Binary configuration file

SEE ALSO

devbuild(1M), mkvtoc(1M), newfs(1M), prtvtoc(1M), vtoc(7), cmpt(7), zdformat(8)

DIAGNOSTICS

Errors during normal operation are signaled to the driver via completion codes in the command block or by controller error interrupts. In response to an error, the driver makes every reasonable attempt to complete the I/O request and to avoid panicking the system. Therefore, the driver will choose to shut down a DCC channel or entire DCC controller rather than panic the system. To shut down a DCC channel or controller means that all the drives on the DCC channel or controller are marked as bad and all the queued I/O requests are failed.

NAME

intro – introduction to system standalone utilities

DESCRIPTION

This section outlines procedures for doing standalone system maintenance.

NAME

boot – run a standalone program or bring up the operating system

SYNOPSIS

From the powerup monitor:

on SCED-based systems:
*** b** [*flag*] [*program* [*arg*] ...]

on SSM-based systems:
---> bh [*'[flag]program* [*arg*] ...']

DESCRIPTION

boot brings up the operating system, or loads other standalone programs into memory.

flag is the value of the configuration flag. Standalone programs (including **boot** and the operating system kernel) interpret various bits of the configuration flag. On SCED systems, if *flag* is not specified, a default value is taken from the most recent **wf** powerup monitor command. On SSM systems, if *flag* is not specified as part of the bootstring it defaults to zero. On SSM systems, there must not be any space characters between *flag* and *program*. Additionally the single quotes are required only if the optional *arg* field is provided.

If the lowest bit of the configuration flag is one (that is, if the configuration flag is odd), **boot** will ask for the name of the program to be loaded into memory. Otherwise, the name is taken from the command line, or (if not present) the value specified the last time the bootstring was set from the powerup monitor. For SCED systems, the command for setting the bootstring is *wn0=bootstring* and for SSM systems, the command is *bh os='bootstring'*.

The name of the program is given in standalone file notation. Standalone file notation is represented as:

dv(*un*,*su*)*file*

where *dv* is the name of the device driver, *un* and *su* are the unit and subunit parameters (interpreted by the device driver), and *file* is the name of the file to be run. Supported device drivers for SCED systems are **sd**, **ts**, **xt** and **zd**. Supported device drivers for SSM systems are **wd** and **tm**. In general, for disk drivers, *un* selects among the various drives, *su* selects a particular disk partition, and *file* is a file name relative to the top level directory of the file system located in the selected partition. For tape drivers, *un* selects among the various drives, *su* selects a particular file on the tape (bounded by EOF tape marks), and *file* is unused.

boot loads the program into memory, displays the allocation of **text**, **data**, and **bss** (uninitialized data), and then begins execution of the program. When the program exits, the powerup monitor resumes.

Arguments for the program may follow the program name, as long as they are separated from the program name by at least one space. **boot** makes the entire (uninterpreted) command line available to the invoked program. The invoked program must then parse the command line to determine the arguments.

WARNINGS

When booting from a disk device, **boot** can only load programs from disks which have a valid VTOC at the correct location. Attempts to boot from disks without a valid VTOC or from a disk partition which is not defined in the VTOC will result in a diagnostic message being printed and a return to the powerup monitor.

NAME

CCSformat – standalone embedded SCSI disk formatter

SYNOPSIS

From the powerup monitor:

on SCED-based systems:
*** b 0** *dv*(*un*,*su*)**stand/CCSformat**

on SSM-based systems:
---> bh *dv*(*un*,*su*)**stand/CCSformat**

DESCRIPTION

CCSformat is the standalone embedded SCSI disk formatter. It is used to format both SCED and SSM SCSI disks. Refer to the *Power-Up Monitor Guide* for detailed instructions on how to use **CCSformat** and the powerup monitor.

SEE ALSO

boot(8)

NAME

copy – perform a standalone data transfer

SYNOPSIS

From the powerup monitor:

on SCED-based systems:
*** b 0** *dv*(*un*,*su*)**stand/copy**

on SSM-based systems:
---> bh *dv*(*un*,*su*)**stand/copy**

DESCRIPTION

copy transfers data from one device (or device partition) to another. The name of the program is given in standalone file notation, where *dv* is the name of the device driver, *un* and *su* are the unit and subunit parameters (interpreted by the device driver). These fields form the device name where the file to be run, *stand/copy*, is to be found. **copy** prompts for the two device names, which are specified in standalone file notation described above (also see **boot**(8)). **copy** also prompts for whether verification is selected (if selected, the data will be reread and verified after it is written), the offsets at which the transfer will begin, and the maximum number of blocks to copy. Messages are intended to be self-explanatory.

WARNINGS

When one of the devices is a disk partition, the disk must have a valid VTOC present in the correct location and the partition referenced must be a valid partition in the VTOC. If any of these conditions fail, then **copy** will print a diagnostic and exit.

SEE ALSO

cp (1), mkvtoc (1M), boot (8), prtvtoc (8)

NAME

dump – standalone memory dump

SYNOPSIS

From the powerup monitor:

on SCED-based systems:
*** b 80** *dv*(*un*,*su*)**stand/dump** *dumpdev offset special* [*size*] [**–o**]

or

*** b 80** *dv*(*un*,*su*)**stand/dump –f** *dumplist filename* [**–o**]

on SSM-based systems:
--> bh '88*dv*(*un*,*su*)**stand/dump** *dumpdev offset special* [*size*] [**–o**]'

or

--> bh '88*dv*(*un*,*su*)**stand/dump –f** *dumplist filename* [**–o**]'

DESCRIPTION

The name of the program is given in standalone file notation, where *dv* is the name of the device driver, *un* and *su* are the unit and subunit parameters (interpreted by the device driver). These fields form the device name where the file to be run, */stand/dump*, is to be found. **dump** copies memory to *offset* blocks (specified as a decimal integer) past the beginning of device *dumpdev* (specified in standalone file notation, described above (also see **boot**(8)). *Filename* should be the name of the same device as *dumpdev* expressed as a standard UNIX character special file (usually beginning with */dev/rdsk/*).

The **–f** flag allows multiple dump devices to be specified. The *dumplist* file name must be specified in standalone notation, and *filename* should be the UNIX filename equivalent of the standalone filename *dumplist*. Each line in the *dumplist* file has the format:

dumpdev offset special [*size*]

Each line of the file describes where a section of the dump will be placed.

By default, all of memory is dumped, up to the size of the dump partition minus the *offset*. If the *size* parameter is specified, then the amount dumped to the partition is the minima of the partition size (as read from the VTOC) and the *size* parameter. Because of this, the *size* parameter is usually not needed, unless the amount dumped to a dump device should be limited for some reason. See WARNINGS.

dump normally aborts if an attempt is made to overwrite an existing dump. If **–o** is specified, **dump** will proceed with the dump even if a previous dump has not been cleared.

The *wn1=dumpstring* powerup monitor command (for SCED-based systems), and the *bh dumper='dumpstring'* (for SSM-based systems) or the *n1=* option of the **bootflags**(1M) command, can be used to store a permanent dump invocation string, so that operating system can automatically generate dumps. The file */etc/init.d/bootflags* already invokes *bootflags* to do this. A typical invocation of dump looks like:

b 80 sd(0,0)stand/dump sd(0,1) 8000 /dev/rdsk/sd0s1

or

bh '88wd(0,0)stand/dump wd(0,1) 8000 /dev/rdsk/wd0s1'

A typical use of dump specifying multiple dump partitions looks like:

b 80 sd(0,0)stand/dump -f sd(0,0)etc/dumplist /etc/dumplist

An example of /etc/dumplist looks like:

```
sd(0,1) 1000 /dev/rdsk/sd0s1
sd(8,1)    0 /dev/rdsk/sd1s1
zd(0,1)    0 /dev/rdsk/zd0s1
```

WARNINGS

If a disk partition is used as a dump device, then that disk must have a valid VTOC in the correct location, and the specified partition must be a valid partition.

SEE ALSO

boot(8)

NAME

prtvtoc – print the VTOC of a standalone disk device

SYNOPSIS

From the powerup monitor:

on SCED-based systems:
*** b 0** *dv(un,su)***stand/prtvtoc**

on SSM-based systems:
--> bh *dv(un,su)***stand/prtvtoc**

DESCRIPTION

The name of the program is given in standalone file notation, where *dv* is the name of the device driver, *un* and *su* are the unit and subunit parameters (interpreted by the device driver). These fields form the device name where the file to be run, *stand/prtvtoc*, is to be found.

prtvtoc allows the contents of the VTOC (volume table of contents) to be viewed in standalone mode for reference or verification. In addition to printing the contents of the VTOC, the command lists any free areas on the disk (that is, those without a partition).

prtvtoc will prompt for a disk name from which to read the VTOC. The device name must be in standalone notation described above (also see **boot**(8)). After the VTOC is printed, another device may be entered. To exit the command, type the word "exit" at the prompt.

See **prtvtoc**(1M) for examples of output.

If the *device* passed to **prtvtoc** does not contain a VTOC, an error will occur. A valid VTOC with a bad checksum can be printed using **prtvtoc** to facilitate recovery of a corrupted VTOC.

SEE ALSO

mkvtoc(1M), prtvtoc(1M), boot(8)

NAME

zdformat – standalone disk formatter for dual-channel disk controller

SYNOPSIS

From the powerup monitor:
*** b 0** *dv*(*un*,*su*)**stand/zdformat**

DESCRIPTION

zdformat is the standalone disk formatter for disks connected to Sequent's dual-channel disk controller. See the *Power-Up Monitor Guide* for detailed instructions on how to use **zdformat.**

SEE ALSO

boot(8)