IMPORTANT NOTICE TO USERS

While every effort has been to ensure the accuracy of all information in this document, AT&T assumes no liability to any party for any loss or damage caused by errors or omissions or statements of any kind in the UNIX® System V Programmer's Reference Manual ©AT&T, its upgrades, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accident or any other cause. AT&T further assumes no liability arising out of the application or use of any product or system described herein; nor any liability for incidental or consequential damages arising from the use of this document. AT&T disclaims all warranties regarding the information contained herein, whether expressed, implied or statutory, including implied warranties or merchantability or fitness for a particular purpose.

AT&T reserves the right to make changes without further notice to any products herein to improve reliability, function or design.

No part of this publication may be reproduced, transmitted or used in any form or by any means—graphic, electronic, mechanical or chemical, including photocopying, recording in any medium, taping, by any computer or information storage and retrieval systems, etc. without prior permission in writing from AT&T.

ACT is a trademark of Micro-Term.
Ann Arbor is a trademark of Ann Arbor Terminals.
Beehive is a trademark of Beehive International Concept
DOCUMENTER'S WORKBENCH is a trademark of AT&T.

HP is a trademark of Hewlett-Packard.
LSI is a trademark of Lear Siegler.
Micro-Term and MIME are trademarks of Micro-Term.
POP is a trademark of Digital Equipment.
Tektronix is a registered trademark of Tektronix.
Versatec is a registered trademark of Versatec.
TeleVideo is a registered trademark of TeleVideo Systems.

UNIX is a registered trademark of AT&T
VAX is a trademark of Digital Equipment

Printed in the United States of America

ISBN 0-13-940479-1 025
1. Commands  1
intro(1) ...................................................... introduction to programming commands
admin(1) ..................................................... create and administer SCCS files
ar(1) ......................................................... archive and library maintainer for portable archives
as(1) .............................................................. common assembler
cb(1) ................................................................. C program beautifier
cc(1) ................................................................. C compiler
cdc(1) ................................................................. change the delta commentary of an SCCS delta
cflow(1) .............................................................. generate C flowgraph
comb(1) .............................................................. combine SCCS deltas
conv(1) .............................................................. common object file converter
convert(1) ........................................................... convert archive files to common formats
cpp(1) .............................................................. the C language preprocessor
cprs(1) .............................................................. compress a common object file
ctrace(1) ........................................................... C program debugger
cxref(1) ........................................................... generate C program cross-reference
delta(1) ........................................................... make a delta (change) to an SCCS file
dis(1) ............................................................... object code disassembler
dump(1) .............................................................. dump selected parts of an object file
gencc(1M) ...................................................... create a front-end to the cc command
get(1) .............................................................. get a version of an SCCS file
infocmp(1M) ..................................................... compare or print out terminfo descriptions
install(1M) ..................................................... install commands
ld(1) ............................................................... link editor for common object files
lex(1) ............................................................. generate programs for simple lexical tasks
lint(1) ............................................................. a C program checker
list(1) ............................................................. produce C source listing from a common object file
lorder(1) .......................................................... find ordering relation for an object library
m4(1) .............................................................. macro processor
make(1) ...................................................... maintain, update, and regenerate groups of programs
mcs(1) ............................................................ manipulate the object file comment section
mkshlib(1) ..................................................... create a shared library
nm(1) ............................................................ print name list of common object file
prof(1) .......................................................... display profile data
prs(1) ............................................................ print an SCCS file
regcmp(1) .................................................... regular expression compile
relogin(1M) .................................................... rename login entry to show current layer
rmdel(1) .......................................................... remove a delta from an SCCS file
# Table of Contents

- `sact(1)` .......................................................... print current SCCS file editing activity
- `sccsdiff(1)` ...................................................... compare two versions of an SCCS file
- `sdb(1)` ............................................................ symbolic debugger
- `size(1)` ............................................................ print section sizes in bytes of common object files
- `strip(1)` ......................................................... strip symbol and line number information from a common object file
- `tic(1M)` ................................................................. terminfo compiler
- `tsort(1)` ........................................................... topological sort
- `unget(1)` .............................................................. undo a previous get of an SCCS file
- `val(1)` ............................................................. validate SCCS file
- `vc(1)` ............................................................... version control
- `what(1)` ............................................................ identify SCCS files
- `wtinit(1M)` ...................................................... object downloader for the 5620 DMD terminal
- `xtd(1M)` .......................................................... extract and print xt driver link structure
- `xts(1M)` .......................................................... extract and print xt driver statistics
- `xtt(1M)` .......................................................... extract and print xt driver packet traces
- `yacc(1)` ............................................................ yet another compiler-compiler

## 2. System Calls 124

- `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)` ............................................................ 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
- `exec(2)` ........................................................ execute a file
- `exit(2)` .......................................................... terminate process
- `fcntl(2)` ........................................................ file control
- `fork(2)` ........................................................ create a new process
- `getdents(2)` .................................................. read directory entries and put in a file
- `getmsg(2)` ........................................................ get next message off a stream
- `getpid(2)` ........................................................ get process, process group, and parent process IDs
- `getuid(2)` ........................................................ 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
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lseek(2)</td>
<td>move read/write file pointer</td>
</tr>
<tr>
<td>mkdir(2)</td>
<td>make a directory</td>
</tr>
<tr>
<td>mknod(2)</td>
<td>make a directory, or a special or ordinary file</td>
</tr>
<tr>
<td>mount(2)</td>
<td>mount a file system</td>
</tr>
<tr>
<td>msgctl(2)</td>
<td>message control operations</td>
</tr>
<tr>
<td>msgget(2)</td>
<td>get message queue</td>
</tr>
<tr>
<td>msgop(2)</td>
<td>message operations</td>
</tr>
<tr>
<td>nice(2)</td>
<td>change priority of a process</td>
</tr>
<tr>
<td>open(2)</td>
<td>open for reading or writing</td>
</tr>
<tr>
<td>pause(2)</td>
<td>suspend process until signal</td>
</tr>
<tr>
<td>pipe(2)</td>
<td>create an interprocess channel</td>
</tr>
<tr>
<td>plock(2)</td>
<td>lock process, text, or data in memory</td>
</tr>
<tr>
<td>poll(2)</td>
<td>STREAMS input/output multiplexing</td>
</tr>
<tr>
<td>profil(2)</td>
<td>execution time profile</td>
</tr>
<tr>
<td>ptrace(2)</td>
<td>process trace</td>
</tr>
<tr>
<td>putmsg(2)</td>
<td>send a message on a stream</td>
</tr>
<tr>
<td>read(2)</td>
<td>read from file</td>
</tr>
<tr>
<td>rmdir(2)</td>
<td>remove a directory</td>
</tr>
<tr>
<td>semctl(2)</td>
<td>semaphore control operations</td>
</tr>
<tr>
<td>semget(2)</td>
<td>get set of semaphores</td>
</tr>
<tr>
<td>semop(2)</td>
<td>semaphore operations</td>
</tr>
<tr>
<td>setpgid(2)</td>
<td>set process group ID</td>
</tr>
<tr>
<td>setuid(2)</td>
<td>set user and group IDs</td>
</tr>
<tr>
<td>shmctl(2)</td>
<td>shared memory control operations</td>
</tr>
<tr>
<td>shmget(2)</td>
<td>get shared memory segment identifier</td>
</tr>
<tr>
<td>shmdt(2)</td>
<td>shared memory operations</td>
</tr>
<tr>
<td>signal(2)</td>
<td>specify what to do upon receipt of a signal</td>
</tr>
<tr>
<td>sigset(2)</td>
<td>signal management</td>
</tr>
<tr>
<td>stat(2)</td>
<td>get file status</td>
</tr>
<tr>
<td>statvfs(2)</td>
<td>get file system information</td>
</tr>
<tr>
<td>stime(2)</td>
<td>set time</td>
</tr>
<tr>
<td>sync(2)</td>
<td>update super block</td>
</tr>
<tr>
<td>sys3b(2)</td>
<td>machine specific functions</td>
</tr>
<tr>
<td>sysfs(2)</td>
<td>get file system type information</td>
</tr>
<tr>
<td>time(2)</td>
<td>get time</td>
</tr>
<tr>
<td>times(2)</td>
<td>get process and child process times</td>
</tr>
<tr>
<td>uadmin(2)</td>
<td>administrative control</td>
</tr>
<tr>
<td>ulimit(2)</td>
<td>get and set user limits</td>
</tr>
<tr>
<td>umask(2)</td>
<td>set and get file creation mask</td>
</tr>
<tr>
<td>umount(2)</td>
<td>unmount a file system</td>
</tr>
<tr>
<td>uname(2)</td>
<td>get name of current UNIX system</td>
</tr>
</tbody>
</table>

**TABLE OF CONTENTS**

vii
Table of Contents

unlink(2) .................................................................................................................................................. remove directory entry
ustat(2) .................................................................................................................................................. get file system statistics
utime(2) .................................................................................................................................................. set file access and modification times
wait(2) .................................................................................................................................................. wait for child process to stop or terminate
write(2) .................................................................................................................................................. write on a file

3. Subroutines 252
intro(3) ............................................................................................................................................... introduction to functions and libraries
a64l(3C) ............................................................................................................................................... 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) ............................................................................................................................................... translate characters
crypt(3C) ............................................................................................................................................... generate hashing encryption
ctermid(3S) ........................................................................................................................................... generate file name for terminal
ctime(3C) ............................................................................................................................................... convert date and time to string
ctype(3C) ............................................................................................................................................... classify characters
cuserid(3S) ........................................................................................................................................... get character login name of the user
dial(3C) ............................................................................................................................................... establish an out-going terminal line connection
drand48(3C) ........................................................................................................................................ generate uniformly distributed pseudo-random numbers
dup2(3C) ............................................................................................................................................... duplicate an open file descriptor
ecvt(3C) ............................................................................................................................................... convert floating-point number to string
eoend(3C) ............................................................................................................................................... last locations in program
close(3S) ............................................................................................................................................... close or flush a stream
ferror(3S) ............................................................................................................................................... stream status inquiries
fopen(3S) ............................................................................................................................................... open a stream
fpgetround(3C) ........................................................................................................................................... IEEE floating point environment control
fread(3S) ............................................................................................................................................... binary input/output
frexp(3C) ............................................................................................................................................... manipulate parts of floating-point numbers
fseek(3S) ............................................................................................................................................... reposition a file pointer in a stream
ftw(3C) ............................................................................................................................................... walk a file tree
getc(3S) ............................................................................................................................................... get character or word from a stream
getcwd(3C) ............................................................................................................................................... get path-name of current working directory
getenv(3C) ............................................................................................................................................... return value for environment name
getgrent(3C) ........................................................................................................................................... get group file entry
cuserid(3S) ............................................................................................................................................... get login name
getopt(3C) ............................................................................................................................................... get option letter from argument vector
getopt(3C) ............................................................................................................................................... get password file entry
Table of Contents

gets(3S) ................................................................. get a string from a stream
getut(3C) ................................................................ access utmp file entry
hsearch(3C) .................................................. manage hash search tables
isnan(3C) ........................................................... test for floating point NaN (Not-A-Number)
l3tol(3C) .................................................. convert between 3-byte integers and long integers
lockf(3C) .......................................................................................................................... record locking on files
lsearch(3C) ......................................................... linear search and update
malloc(3C) ........................................................................ main memory allocator
memory(3C) .................................................................................................................. memory operations
mktemp(3C) .................................................................................................. make a unique file name
monitor(3C) ........................................................................................ prepare execution profile
nlist(3C) ............................................................... get entries from name list
perror(3C) ........................................................................ system error messages
popen(3S) ................................................................................................ initiate pipe to/from a process
printf(3S) ........................................................................................................ print formatted output
putc(3S) ................................................................................................ put character or word on a stream
putenv(3C) ................................................................................................................................ change or add value to environment
putpwent(3C) ................................................................................... write password file entry
puts(3S) .................................................................................................................. put a string on a stream
qsort(3C) .................................................................................................................. quicker sort
rand(3C) .................................................................................................................. simple random-number generator
scanf(3S) ................................................................................................ convert formatted input
setbuf(3S) .................................................................................................................. assign buffering to a stream
setjmp(3C) .................................................................................................................. non-local goto
sleep(3C) ........................................................................................................ suspend execution for interval
ssignal(3C) ............................................................................................................... software signals
stdio(3S) .................................................................................................................. standard buffered input/output package
stdlibc(3C) ........................................................................................................ standard interprocess communication package
string(3C) ........................................................................................................ string operations
strtod(3C) ........................................................................................................ convert string to double-precision number
strtol(3C) ........................................................................................................ convert string to integer
swab(3C) .................................................................................................................. swap bytes
system(3S) .................................................................................................. issue a shell command
tmpfile(3S) ................................................................................................ create a temporary file
tmpnam(3S) ................................................................................................ create a name for a temporary file
tsearch(3C) ................................................................................................ manage binary search trees
ttyname(3C) ................................................................................................ find name of a terminal
ttyslot(3C) ................................................................................................ find the slot in the utmp file of the current user
ungetc(3S) ................................................................................................ push character back into input stream
vprintf(3S) ................................................................................................ print formatted output of a varargs argument list
bessel(3M) ................................................................................................ Bessel functions
Table of Contents

erf(3M) ....................................................... error function and complementary error function
exp(3M) .............................................. exponential, logarithm, power, square root functions
floor(3M) .................................................. floor, ceiling, remainder, absolute value functions
gamma(3M) .......................................................... log gamma function
hypot(3M) .......................................................... Euclidean distance function
matherr(3M) .......................................................... error-handling function
sinh(3M) .......................................................... hyperbolic functions
trig(3M) .......................................................... trigonometric functions
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
assert(3X) .......................................................... verify program assertion
crypt(3X) .......................................................... password and file encryption functions
curses(3X) ................................................. terminal screen handling and optimization package
directory(3X) .......................................................... directory operations
Idahread(3X) .......................................................... read the archive header of a member of an archive file
Idclose(3X) .......................................................... close a common object file
ldfhdr(3X) .......................................................... read the file header of a common object file
ldgetname(3X) .......................................................... retrieve symbol name for common object file symbol table entry

x  PROGRAMMER'S REFERENCE MANUAL
ldlread(3X) ........................ manipulate line number entries of a common object file
ldlseek(3X) ............................ 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) ..................................... open a common object file for reading
ldrseek(3X) ................................. seek to relocation entries of a section of a common object file
ldshread(3X) .............................. read an indexed/named section header of a common object file
ldsseek(3X) ................................. seek to an indexed/named section of a common object file
ldtbindx(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
libwindows(3X) ................................. windowing terminal function library
logname(3X) ........................................................ return login name of user
malloc(3X) ........................................................ fast main memory allocator
plot(3X) .................................................. graphics interface subroutines
regcmp(3X) ........................................................ compile and execute regular expression
abort(3F) ........................................................... terminate Fortran program
abs(3F) ........................................................ Fortran absolute value
acos(3F) ........................................ Fortran arccosine intrinsic function
aimag(3F) ........................................ Fortran imaginary part of complex argument
aint(3F) ........................................ Fortran integer part intrinsic function
asin(3F) ........................................ Fortran arcsine intrinsic function
atan(3F) .......................................... Fortran arctangent intrinsic function
atan2(3F) ........................................ Fortran arctangent intrinsic function
bool(3F) ........................................ Fortran Bitwise Boolean functions
conjg(3F) .......................................... Fortran complex conjugate intrinsic function
cos(3F) ........................................ Fortran cosine intrinsic function
cosh(3F) ........................................ Fortran hyperbolic cosine intrinsic function
dim(3F) ........................................ positive difference intrinsic functions
dprod(3F) ........................................ double precision product intrinsic function
exp(3F) .......................................... Fortran exponential intrinsic function
ftype(3F) ........................................ explicit Fortran type conversion
getarg(3F) ........................................ return Fortran command-line argument
getenv(3F) ....................................... return Fortran environment variable
iargc(3F) ........................................ return the number of command line arguments
index(3F) ........................................ return location of Fortran substring
len(3F) ........................................ return length of Fortran string
log(3F) ........................................ Fortran natural logarithm intrinsic function
log10(3F) ....................................... Fortran common logarithm intrinsic function
max(3F) ........................................ Fortran maximum-value functions
mclock(3F) ..................................... return Fortran time accounting
mil(3F) ........................................ Fortran Military Standard functions
Table of Contents

min(3F) ................................................................. Fortran minimum-value functions
mod(3F) ................................................................. Fortran remaindering intrinsic functions
rand(3F) ................................................................. random number generator
round(3F) ................................................................. Fortran nearest integer functions
sign(3F) ................................................................. Fortran transfer-of-sign intrinsic functions
signal(3F) ............................................................... specify Fortran action on receipt of a system signal
sin(3F) ................................................................. Fortran sine intrinsic function
sinh(3F) ................................................................. Fortran hyperbolic sine intrinsic function
sqrt(3F) ................................................................. Fortran square root intrinsic function
strcsp(3F) ............................................................... string comparison intrinsic functions
system(3F) ............................................................. issue a shell command from Fortran
tan(3F) ................................................................. Fortran tangent intrinsic function
tanh(3F) ................................................................. Fortran hyperbolic tangent intrinsic function

4. File Formats 515
intro(4) ................................................................. introduction to file formats
a.out(4) ................................................................. common assembler and link editor output
acct(4) ................................................................. per-process accounting file format
ar(4) ................................................................. common archive file format
checklist(4) ........................................................ list of file systems processed by fsck and ncheck
core(4) ................................................................. format of core image file
cpio(4) ................................................................. format of cpio archive
dir(4) ................................................................. format of directories
dirent(4) ................................................................. file system independent directory entry
.edt_swapp(4) ........................................................ software application file
filehdr(4) ............................................................. file header for common object files
fs(4) ................................................................. format of system volume
fsubc(4) ................................................................. format specification in text files
fstab(4) ................................................................. file-system-table
gettydefs(4) ........................................................ speed and terminal settings used by getty
gps(4) ................................................................. graphical primitive string, format of graphical files
group(4) ................................................................. group file
inittab(4) ............................................................... script for the init process
inode(4) ................................................................. format of an i-node
issue(4) ................................................................. issue identification file
ldfncn(4) ............................................................... common object file access routines
limits(4) ............................................................... file header for implementation-specific constants
linenum(4) ............................................................ line number entries in a common object file
master(4) ............................................................. master configuration database
mnttab(4) ............................................................. mounted file system table
passwd(4) ............................................................. password file
plot(4) .......................................................... graphics interface
pnch(4) .......................................................... file format for card images
profile(4) ..................................................... setting up an environment at login time
reloc(4) ....................................................... relocation information for a common object file
rfmaster(4) ................................................ Remote File Sharing name server master file
scsfile(4) .................................................... format of SCCS file
scnhdr(4) .................................................... section header for a common object file
scr_dump(4) ................................................ format of curses screen image file
syms(4) ........................................................ common object file symbol table format
system(4) ..................................................... system configuration information table
term(4) ........................................................ format of compiled term file
terminfo(4) ................................................ terminal capability data base
timezone(4) ................................................ set default system time zone
unistd(4) ..................................................... file header for symbolic constants
utmp(4) ....................................................... utmp and wtmp entry formats

5. Miscellaneous Facilities 609
intro(5) ...................................................... introduction to miscellany
ascii(5) ...................................................... map of ASCII character set
environ(5) .................................................. user environment
fcntl(5) ..................................................... file control options
jagent(5) .................................................... host control of windowing terminal
layers(5) ................................................... protocol used between host and windowing terminal under
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
xtproto(5) ............................................... multiplexed channels protocol used by xt(7) driver
Introduction

This manual describes the programming features of the UNIX system. It provides neither a general overview of the UNIX system nor details of the implementation of the system.

Not all commands, features, and facilities described in this manual are available in every UNIX system. Some of the features require additional utilities which may not exist on your system.

This manual is divided into five sections, some containing interfiled subclasses:

1. Commands
2. System Calls.
3. Subroutines:
   3C. C Programming Language Libraries
   3S. Standard I/O Library Routines
   3M. Mathematical Library Routines
   3N. Networking Support Utilities
   3X. Specialized Libraries
   3F. FORTRAN Programming Libraries
4. File Formats.
5. Miscellaneous Facilities.

Section 1 (Commands) describes commands that support C and other programming languages.

Section 2 (System Calls) describes the access to the services provided by the UNIX 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. See intro(3) for descriptions of 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 things. Included are descriptions of character sets, macro packages, etc.

References with numbers other than those above mean that the utility is contained in the appropriate section of another manual. References with (1) following the command mean that the utility is contained in this manual or the User's Reference Manual (P-H). Those followed by (1M), (7), or (8) are contained in the System Administrator's Reference Manual (AT&T).

Each section consists of a number of independent entries of a page or so each. Entries within each section are alphabetized, with the exception of the introductory entry that begins each section (also Section 3 is in alphabetical order by suffixes). Some entries may describe several routines, commands, etc. 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.

All entries are based on a common format, not all of whose parts always appear:

- The NAME part gives the name(s) of the entry and briefly states its purpose.

- The SYNOPSIS part summarizes the use of the program being described. A few conventions are used, particularly in Section 2 (System Calls):
  - 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 always refers to a file name.
  - 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 =.
The DESCRIPTION part describes the utility.

The EXAMPLE(S) part gives example(s) of usage, where appropriate.

The FILES part gives the file names that are built into the program.

The SEE ALSO part gives pointers to related information.

The DIAGNOSTICS part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The NOTES part gives generally "helpful hints" about the use of the utility.

The WARNINGS part points out potential pitfalls.

The BUGS part gives known bugs and sometimes deficiencies.

The CAVEATS part gives details of the implementation that might affect usage.

A "Table of Contents" and a "Permuted Index" derived from that table precedes section 1.

Some of the manual pages refer to the MAU, which is the WE®32106 Math Acceleration Unit. This unit is a high performance floating point support processor chip that supports a large subset of IEEE (Institute of Electrical and Electronics Engineers) P754 Draft 10 requirements for Binary Floating Point Arithmetic.
UNIX® System V

INTRO(1)

NAME
intro – introduction to programming commands

DESCRIPTION
This section describes, in alphabetical order, commands available for the AT&T 3B2 Computer. The top of each page indicates the utilities package to which the command belongs. The packages are:

Advanced C Utilities
AT&T Windowing Utilities
C Programming Language Utilities
Directory and File Management Utilities
Extended Software Generation System Utilities
Software Generation System Utilities
Source Code Control System Utilities
Terminal Information Utilities

COMMAND SYNTAX
Unless otherwise noted, the commands described accept options and other arguments according to the following syntax:

name [option(s)] [cmdarg(s)]

where:
name is the name of an executable file
option is – noargletter(s) or
– argletter < > optarg

where:
noargletter is a single letter representing an option without an option-argument
argletter is a single letter representing an option requiring an option-argument
<> is optional white space
optarg is an option-argument (character string) satisfying the preceding argletter.

cmdarg is a path name (or other command argument) not beginning with “–”, or “_” by itself indicating the standard input.

Throughout the manual pages there are references to TMPDIR, BINDIR, INCDIR, LIBDIR, and LLIBDIR. These represent directory names whose value is specified on each manual page as necessary. For example, TMPDIR might refer to /tmp or /usr/tmp. These are not environment variables and cannot be set. [There is also an environment variable called TMPDIR which can be set. See tmpnam(3S).]
SEE ALSO
exit(2), wait(2), getopt(3C).
getopts(1) in the User's Reference Manual,

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

WARNINGS
Some commands produce unexpected results when processing files containing
null characters. These commands often treat text input lines as strings and
therefore become confused upon encountering a null character (the string termi-
nator) within a line.
NAME
admin - create and administer SCCS files

SYNOPSIS
admin [-n] [-i[name]] [-r[rel]] [-t[name]] [-f[flag[flag-val]]] [-d[flag[flag-val]]]
[-a[login]] [-e[login]] [-m[mrlist]] [-y[comment]] [-h] [-z] files

DESCRIPTION
admin is used to create new SCCS files and change parameters of existing ones. Arguments to
admin, which may appear in any order, consist of keyletter arguments, which begin with -,
and named files (note that SCCS file names must begin with the characters s.). If a named file
does not exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a
named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, admin behaves as though each file in the directory were
specified as a named file, except that non-SCCS files (last component of the path
name does not begin with s.) and unreadable files are silently ignored. If a
name of - is given, the standard input is read; each line of the standard input is
taken to be the name of an SCCS file to be processed. Again, non-SCCS files and
unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one
named file is to be processed since the effects of the arguments apply independ­ently
to each named file.

-n This keyletter indicates that a new SCCS file is to be created.

-i[name] The name of a file from which the text for a new SCCS file
is to be taken. The text constitutes the first delta of the
file (see -r keyletter for delta numbering scheme). If the i
keyletter is used, but the file name is omitted, the text is
obtained by reading the standard input until an end-of-file
is encountered. If this keyletter is omitted, then the SCCS
file is created empty. Only one SCCS file may be created
by an admin command on which the i keyletter is sup­plied. Using a single admin to create two or more SCCS
files requires that they be created empty (no -i keyletter).
Note that the -i keyletter implies the -n keyletter.

-r[rel] The release into which the initial delta is inserted. This
keyletter may be used only if the -i keyletter is also used.
If the -r keyletter is not used, the initial delta is inserted
into release 1. The level of the initial delta is always 1 (by
default initial deltas are named 1.1).

-t[name] The name of a file from which descriptive text for the
SCCS file is to be taken. If the -t keyletter is used and
admin is creating a new SCCS file (the -n and/or -i
keyletters also used), the descriptive text file name must
also be supplied. In the case of existing SCCS files: (1) a
The &lt;list&gt; has the following syntax:

\[
\text{<list>} := \text{<range>} \mid \text{<list>}, \text{<range>}
\]

The character &lt;a&gt; in the &lt;list&gt; is equivalent to specifying all releases for the named SCCS file.

Causes &lt;delta(1)&gt; to create a “null” delta in each of those releases (if any) being skipped when a delta is made in a new release (e.g., in making &lt;delta 5.1&gt; after &lt;delta 2.7&gt;, releases 3 and 4 are skipped). These null deltas serve as “anchor points” so that branch deltas may later be created from them. The absence of this flag causes skipped
releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

**qtext**  
User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by get(1).

**mmod**  
Module name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by get(1). If the m flag is not specified, the value assigned is the name of the SCCS file with the leading s removed.

**ttype**  
Type of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by get(1).

**vpgm**  
Causes delta(1) to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validity checking program [see delta(1)]. (If this flag is set when creating an SCCS file, the m keyletter must also be used even if its value is null).

**-dflag**  
Causes removal (deletion) of the specified flag from an SCCS file. The -d keyletter may be specified only when processing existing SCCS files. Several -d keyletters may be supplied on a single admin command. See the -f keyletter for allowable flag names.

**list**  
A list of releases to be "unlocked". See the -f keyletter for a description of the I flag and the syntax of a list.

**-alogin**  
A login name, or numerical UNIX system group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all login names common to that group ID. Several a keyletters may be used on a single admin command line. As many logins, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If login or group ID is preceded by a ! they are to be denied permission to make deltas.

**-elogin**  
A login name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all login names common to that group ID. Several e keyletters may be used on a single admin command line.
The list of Modification Requests (MR) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to `delta(1)`. The v flag must be set and the MR numbers are validated if the v flag has a value (the name of an MR number validation program). Diagnostics will occur if the v flag is not set or MR validation fails.

The comment text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of `delta(1)`. Omission of the -y keyletter results in a default comment line being inserted in the form:

date and time created YY/MM/DD HH:MM:SS by login

The -y keyletter is valid only if the -i and/or -n keyletters are specified (i.e., a new SCCS file is being created).

Causes admin to check the structure of the SCCS file [see `sccsfile(5)`], and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced. keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see -h, above).

Note that use of this keyletter on a truly corrupted file may prevent future detection of the corruption.

The last component of all SCCS file names must be of the form s.file-name. New SCCS files are given mode 444 [see `chmod(1)`]. Write permission in the pertinent directory is, of course, required to create a file. All writing done by admin is to a temporary x-file, called x.file-name, [see `get(1)`], created with mode 444 if the admin command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of admin, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of `ed(1)`. Care must be taken! The edited file should always be processed by an admin -h to check for corruption followed by an admin -z to generate a proper check-sum.
Another `admin -h` is recommended to ensure the SCCS file is valid.

`admin` also makes use of a transient lock file (called `z.file-name`), which is used to prevent simultaneous updates to the SCCS file by different users. See `get(1)` for further information.

### FILES

- **g-file**: Existed before the execution of `delta`; removed after completion of `delta`.
- **p-file**: Existed before the execution of `delta`; may exist after completion of `delta`.
- **q-file**: Created during the execution of `delta`; removed after completion of `delta`.
- **x-file**: Created during the execution of `delta`; renamed to SCCS file after completion of `delta`.
- **z-file**: Created during the execution of `delta`; removed during the execution of `delta`.
- **d-file**: Created during the execution of `delta`; removed after completion of `delta`.

- **/usr/bin/bdiff**: Program to compute differences between the "gotten" file and the `g-file`.

### SEE ALSO


### DIAGNOSTICS

Use `help(1)` for explanations.
NAME
ar — archive and library maintainer for portable archives

SYNOPSIS
ar key [posname] afile [name] ...

DESCRIPTION
The ar command maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose. The magic string and the file headers used by ar consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

When ar creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure is described in detail in ar(4). The archive symbol table [described in ar(4)] is used by the link editor [ld(1)] to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by ar when there is at least one object file in the archive. The archive symbol table is in a specially named file which is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the ar(1) command is used to create or update the contents of such an archive, the symbol table is rebuilt. The s option described below will force the symbol table to be rebuilt.

Unlike command options, the command key is a required part of ar's command line. The key (which may begin with a -) is formed with one of the following letters: dqrtpmx. Arguments to the key, alternatively, are made with one or more of the following set: vuaibcls. Posname is an archive member name used as a reference point in positioning other files in the archive. Afile is the archive file. The names are constituent files in the archive file. The meanings of the key characters are as follows:

d    Delete the named files from the archive file.

r    Replace the named files in the archive file. If the optional character u is used with r, then only those files with dates of modification later than the archive files are replaced. If an optional positioning character from the set ab is used, then the posname argument must be present and specifies that new files are to be placed after (a) or before (b or i) posname. Otherwise new files are placed at the end.

q    Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. This option is useful to avoid quadratic behavior when creating a large archive piece-by-piece. Unchecked, the file may grow exponentially up to the second degree.

t    Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p    Print the named files in the archive.
m  Move the named files to the end of the archive. If a positioning character is present, then the `posname` argument must be present and, as in r, specifies where the files are to be moved.

x  Extract the named files. If no names are given, all files in the archive are extracted. In neither case does x alter the archive file.

The meanings of the key arguments are as follows:

v  Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with t, give a long listing of all information about the files. When used with x, precede each file with a name.

c  Suppress the message that is produced by default when afile is created.

l  Place temporary files in the local (current working) directory rather than in the default temporary directory, TMPDIR.

s  Force the regeneration of the archive symbol table even if ar(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the strip(1) command has been used on the archive.

FILES

$TMPDIR/*  temporary files

$TMPDIR is usually /usr/tmp but can be redefined by setting the environment variable TMPDIR [see tempnam() in tmpnam(3S)].

SEE ALSO

ld(1), lorder(1), strip(1), tmpnam(3S), a.out(4), ar(4)

NOTES

If the same file is mentioned twice in an argument list, it may be put in the archive twice.
NAME
as - common assembler

SYNOPSIS
as [options] filename

DESCRIPTION
The as command assembles the named file. The following flags may be
specified in any order:
-o objfile  Put the output of the assembly in objfile. By default, the output file
name is formed by removing the .s suffix, if there is one, from the
input file name and appending a .o suffix.
-n  Turn off long/short address optimization. By default, address optimi-
   zation takes place.
-m  Run the m4 macro processor on the input to the assembler.
-R  Remove (unlink) the input file after assembly is completed.
-dl  Do not produce line number information in the object file.
-V  Write the version number of the assembler being run on the standard
    error output.
-Y [md],dir  Find the m4 preprocessor (m) and/or the file of predefined macros (d)
    in directory dir instead of in the customary place.

FILES
TMPDIR/*       temporary files

TMPDIR is usually /usr/tmp but can be redefined by setting the environment
variable TMPDIR [see tempnam() in tmpnam(3S)].

SEE ALSO
cc(1), ld(1), m4(1), nm(1), strip(1), tmpnam(3S), a.out(4)

WARNING
If the -m (m4 macro processor invocation) option is used, keywords for m4 [see
m4(1)] cannot be used as symbols (variables, functions, labels) in the input file
since m4 cannot determine which are assembler symbols and which are real m4
macros.

BUGS
The .align assembler directive may not work in the .text section when optimiza-

CAVEATS
Arithmetic expressions may only have one forward referenced symbol per
expression.

NOTES
Wherever possible, the assembler should be accessed through a compilation
system interface program [such as cc(1)].
NAME
cb - C program beautifier

SYNOPSIS
   cb [ -s ] [ -j ] [ -l leng ] [ file ... ]

DESCRIPTION
   The cb command reads C programs either from its arguments or from the standard
   input, and writes them on the standard output with spacing and indentation that
   display the structure of the code. Under default options, cb preserves all user
   new-lines.

   cb accepts the following options.
   -s   Canonicalizes the code to the style of Kernighan and Ritchie in The C
        Programming Language.
   -j   Causes split lines to be put back together.
   -l leng   Causes cb to split lines that are longer than leng.

SEE ALSO
   cc(1),

BUGS
   Punctuation that is hidden in preprocessor statements will cause indentation
   errors.
NAME
  cc - C compiler

SYNOPSIS
  cc [ options ] files

DESCRIPTION
  The cc command is the interface to the C Compilation System. The compilation
  tools consist of a preprocessor, compiler, optimizer, assembler and link editor.
  The cc command processes the supplied options and then executes the various
  tools with the proper arguments. The cc command accepts several types of files
  as arguments:

  Files whose names end with .c are taken to be C source programs and may be
  preprocessed, compiled, optimized, assembled and link edited. The compilation
  process may be stopped after the completion of any pass if the appropriate
  options are supplied. If the compilation process runs through the assembler
  then an object program is produced and is left in the file whose name is that of
  the source with .o substituted for .c. However, the .o file is normally deleted if
  a single C program is compiled and then immediately link edited. In the same
  way, files whose names end in .s are taken to be assembly source programs, and
  may be assembled and link edited; and files whose names end in .i are taken to
  be preprocessed C source programs and may be compiled, optimized, assembled
  and link edited. Files whose names do not end in .c, .s or .i are handed to the
  link editor.

  Since the cc command usually creates files in the current directory during the
  compilation process, it is necessary to run the cc command in a directory in
  which a file can be created.

  The following options are interpreted by cc:

  -c Supress the link editing phase of the compilation, and do not remove
        any produced object files.

  -ds Don't generate symbol attribute information for the symbolic debugger.

  -dl Don't generate symbolic debugging line number information. This and
        the above flag may be used in conjunction as -dsl (-dsl is the default
        unless the -g flag is given).

  -f Cause the link editor to load floating point emulation code. This option
        is needed only when loading old objects which use floating point.
        Current objects do not require floating point emulation code.

  -g Cause the compiler to generate additional information needed for the
        use of sdb (1).

  -o outfile
        Produce an output object file by the name outfile. The name of the
        default file is a.out. This is a link editor option.

  -p Arrange for the compiler to produce code that counts the number of
        times each routine is called; also, if link editing takes place, profiled
        versions of libc.a and libm.a (with -lm option) are linked and monitor(3C)
        is automatically called. A mon.out file will then be produced at normal
termination of execution of the object program. An execution profile can
then be generated by use of prof(1).

-qP
  Arrange for profiled code to be produced where the p argument pro-
  duces identical results to the -p option [allows profiling with prof(1)].

-B string
-t[p02al]
  These options will be removed in the next release. Use the -Y option.

-E
  Run only cpp(1) on the named C programs, and send the result to the
  standard output.

-F
  Cause the compiler to generate code for single precision arithmetic
  whenever an expression contains float variables and no doubles.

-H
  Print out on stderr the pathname of each file included during the current
  compilation.

-O
  Do compilation phase optimization. This option will not have any affect
  on .s files.

-P
  Run only cpp(1) on the named C programs and leave the result in
  corresponding files suffixed .i. This option is passed to cpp(1).

-S
  Compile and do not assemble the named C programs, and leave the
  assembler-language output in corresponding files suffixed .s.

-V
  Print the version of the compiler, optimizer, assembler and/or link
  editor that is invoked.

-Wc,arg1[,arg2...]
  Hand off the argument[s] argi to pass c where c is one of [p02al] indi-
  cating the preprocessor, compiler, optimizer, assembler, or link editor,
  respectively. For example: -Wa,-m passes -m to the assembler.

-Y [p02alSILU],dirname
  Specify a new pathname, dirname, for the locations of the tools and
directories designated in the first argument. [p02alSILU] represents:

    p  preprocessor
    0  compiler
    2  optimizer
    a  assembler
    l  link editor
    S  directory containing the start-up routines
    I  default include directory searched by cpp(1)
    L  first default library directory searched by ld(1)
    U  second default library directory searched by ld(1)

  If the location of a tool is being specified, then the new pathname for
  the tool will be dirname/tool. If more than one -Y option is applied to
  any one tool or directory, then the last occurrence holds.

The cc command also recognizes -C, -D, -H, -I and -U and passes these
options and their arguments directly to the preprocessor without using the -W
option. Similarly, the cc command recognizes -a, -l, -m, -o, -r, -s, -t, -u,
-x, -z, -L, -M and -V and passes these options and their arguments directly
to the loader. See the manual pages for `cpp(1)` and `ld(1)` for descriptions.

Other arguments are taken to be C compatible object programs, typically produced by an earlier `cc` run, or perhaps libraries of C compatible routines and are passed directly to the link editor. These programs, together with the results of any compilations specified, are link edited (in the order given) to produce an executable program with name `a.out` unless the `-o` option of the link editor is used.

If the `cc` command is put in a file `prefixcc` the prefix will be parsed off the command and used to call the tools, i.e., `prefixtool`. For example, OLDcc will call OLDcpp, OLDcomp, OLDoptim, OLDas, and OLDld and will link OLDcrt1.o. Therefore, one MUST be careful when moving the `cc` command around. The prefix will apply to the preprocessor, compiler, optimizer, assembler, link editor, and the start-up routines.

The C language standard was extended to allow arbitrary length variable names. The option pair `"-Wp, -T -WO, -XT"` will cause `cc` to truncate arbitrary length variable names.

**FILES**

- `file.c`: C source file
- `file.i`: preprocessed C source file
- `file.o`: object file
- `file.s`: assembly language file
- `a.out`: link edited output
- `LIBDIR/*/crt1.o`: start-up routine
- `LIBDIR/crtin.o`: start-up routine
- `TMPDIR/*`: temporary files
- `LIBDIR/cpp`: preprocessor, `cpp(1)`
- `LIBDIR/comp`: compiler
- `LIBDIR/optim`: optimizer
- `BINDIR/as`: assembler, `as(1)`
- `BINDIR/ld`: link editor, `ld(1)`
- `LIBDIR/libc.a`: standard C library
- `LIBDIR/libc_s.a`: standard C shared library

`LIBDIR` is usually `/lib`

`BINDIR` is usually `/bin`

`TMPDIR` is usually `/usr/tmp` but can be redefined by setting the environment variable `TMPDIR` [see `tempnam()` in `tmpnam(3S)`].

**SEE ALSO**

- `as(1)`, `ld(1)`, `cpp(1)`, `gencc(1)`, `lint(1)`, `prof(1)`, `sdb(1)`, `tmpnam(3S)`.

**DIAGNOSTICS**

The diagnostics produced by the C compiler are sometimes cryptic. Occasional messages may be produced by the assembler or link editor. If an outdated version of the assembler has been installed over the assembler that is part of Issue 4 of the C Programming Language Utilities and that outdated assembler is used with the Issue 4 compiler, one of the following error messages will appear.
aline 2: invalid instruction name
... "version"

aline 2: Invalid instruction name
aline 2: syntax error

To ensure that the compilation system works correctly, re-install the C Programming Language Utilities Issue 4 assembler.

NOTES

By default, the return value from a compiled C program is completely random. The only two guaranteed ways to return a specific value is to explicitly call `exit(2)` or to leave the function `main()` with a "return expression;" construct.
NAME
cdc – change the delta commentary of an SCCS delta

SYNOPSIS
cdc -rSID [-m[mrlist]] [-y[comment]] files

DESCRIPTION
cdc changes the delta commentary, for the SID (SCCS IDENTification string) specified by the -r keyletter, of each named SCCS file.

Delta commentary is defined to be the Modification Request (MR) and comment information normally specified via the delta(1) command (-m and -y keyletters).

If a directory is named, cdc behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with `s`) and unreadable files are silently ignored. If a name of - is given, the standard input is read (see WARNINGS) and each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to cdc, which may appear in any order, consist of keyletter arguments and file names.

All the described keyletter arguments apply independently to each named file:

- **-rSID**
  Used to specify the SCCS IDENTification (SID) string of a delta for which the delta commentary is to be changed.

- **-mmrlist**
  If the SCCS file has the v flag set [see admin(1)] then a list of MR numbers to be added and/or deleted in the delta commentary of the SID specified by the -r keyletter may be supplied. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of delta(1). In order to delete an MR, precede the MR number with the character ! (see EXAMPLES). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a "comment" line. A list of all deleted MRs is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If -m is not used and the standard input is a terminal, the prompt MRs? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The MRs? prompt always precedes the comments? prompt (see -y keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the v flag has a value [see admin(1)], it is taken to be the name of a program (or shell procedure) which validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number
validation program, cdc terminates and the delta commentary remains unchanged.

\[-y\{comment\}\] Arbitrary text used to replace the comment(s) already existing for the delta specified by the \(-r\) keyletter. The previous comments are kept and preceded by a comment line stating that they were changed. A null comment has no effect.

If \(-y\) is not specified and the standard input is a terminal, the prompt \texttt{comments?} is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped newline character terminates the comment text.

Simply stated, the keyletter arguments are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

**EXAMPLES**

\[\texttt{cdc \(-r1.6 \hspace{1em} -m"bl78-12345 !bl77-54321 bl79-00001\" \hspace{1em} -y\text{trouble \hspace{1em} s.file}\hspace{1em}}\]

adds bl78-12345 and bl79-00001 to the MR list, removes bl77-54321 from the MR list, and adds the comment trouble to delta 1.6 of s.file.

\[\texttt{cdc \(-r1.6 \hspace{1em} s.file\hspace{1em} \text{MRs?} \hspace{1em} !bl77-54321 \hspace{1em} bl78-12345 \hspace{1em} bl79-00001\hspace{1em}}\]

\[\texttt{comments? \hspace{1em} trouble\hspace{1em}}\]

\[\text{does the same thing.}\]

**WARNINGS**

If SCCS file names are supplied to the cdc command via the standard input (\(-\) on the command line), then the \(-m\) and \(-y\) keyletters must also be used.

**FILES**

- x-file \[see delta(1)\]
- z-file \[see delta(1)\]

**SEE ALSO**

admin(1), delta(1), get(1), prs(1), sccsfile(4).


**DIAGNOSTICS**

Use help(1) for explanations.
NAME
cflow — generate C flowgraph

SYNOPSIS
cflow [-r] [-ix] [-i_] [-dnum] files

DESCRIPTION
The cflow command analyzes a collection of C, yacc, lex, assembler, and object
files and attempts to build a graph charting the external references. Files
suffixed with .y, .l, and .c are yacced, lexed, and C-preprocessed as appropriate.
The results of the preprocessed files, and files suffixed with .i, are then run
through the first pass of lint(1). Files suffixed with .s are assembled. Assembled
files, and files suffixed with .o, have information extracted from their symbol
tables. The results are collected and turned into a graph of external references
which is displayed upon the standard output.

Each line of output begins with a reference number, followed by a suitable
number of tabs indicating the level, then the name of the global symbol fol­
lowed by a colon and its definition. Normally only function names that do not
begin with an underscore are listed (see the -i options below). For information
extracted from C source, the definition consists of an abstract type declaration
(e.g., char *), and, delimited by angle brackets, the name of the source file and
the line number where the definition was found. Definitions extracted from
object files indicate the file name and location counter under which the symbol
appeared (e.g., text). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that
name contain only the reference number of the line where the definition may be
found. For undefined references, only <> is printed.

As an example, given the following in file.c:

int i;
main() {
    f();
    g();
    f();
}
f()
{
    i = h();
}
the command

cflow –ix file.c

produces the output

1 main: int(), <file.c 4>
2 f: int(), <file.c 11>
3 h: <>
4 i: int, <file.c 1>
5 g: <>

When the nesting level becomes too deep, the output of cflow can be piped to pr(1), using the –e option, to compress the tab expansion to something less than every eight spaces.

In addition to the –D, –I, and –U options [which are interpreted just as they are by cc(1) and cpp(1)], the following options are interpreted by cflow:

–r Reverse the “caller:callee” relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.

–ix Include external and static data symbols. The default is to include only functions in the flowgraph.

–i Include names that begin with an underscore. The default is to exclude these functions (and data if –ix is used).

–dnum The num decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be ignored.

DIAGNOSTICS
Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C-preprocessor).

SEE ALSO
as(1), cc(1), cpp(1), lex(1), lint(1), nm(1), yacc(1).

BUGS
Files produced by lex(1) and yacc(1) cause the reordering of line number declarations which can confuse cflow. To get proper results, feed cflow the yacc or lex input.
NAME
comb — combine SCCS deltas

SYNOPSIS
comb files

DESCRIPTION
comb generates a shell procedure [see sh(1)] which, when run, will reconstruct
the given SCCS files. The reconstructed files will, hopefully, be smaller than the
original files. The arguments may be specified in any order, but all keyletter
arguments apply to all named SCCS files. If a directory is named, comb behaves
as though each file in the directory were specified as a named file, except that
non-SCCS files (last component of the path name does not begin with $s) and
unreadable files are silently ignored. If a name of — is given, the standard input
is read; each line of the input is taken to be the name of an SCCS file to be pro­
cessed; non-SCCS files and unreadable files are silently ignored. The generated
shell procedure is written on the standard output.

The keyletter arguments are as follows. Each is explained as though only one
named file is to be processed, but the effects of any keyletter argument apply
independently to each named file. Each get —e generated, this argument causes
the reconstructed file to be accessed at the release of the delta to be created, oth­
erwise the reconstructed file would be accessed at the most recent ancestor. Use
of the —o keyletter may decrease the size of the reconstructed SCCS file. It may
also alter the shape of the delta tree of the original file. argument causes comb
to generate a shell procedure which, when run, will produce a report giving, for
each file: the file name, size (in blocks) after combining, original size (also in
blocks), and percentage change computed by:

100 • (original — combined) / original

It is recommended that before any SCCS files are actually combined, one should
use this option to determine exactly how much space is saved by the combining
process. SCCS IDentification string (SID) of the oldest delta to be preserved. All
older deltas are discarded in the reconstructed file. list (see get(1) for the syntax
of a list) of deltas to be preserved. All other deltas are discarded.

If no keyletter arguments are specified, comb will preserve only leaf deltas and
the minimal number of ancestors needed to preserve the tree.

FILES
s.COMB The name of the reconstructed SCCS file.
comb???? Temporary.

SEE ALSO
admin(1), delta(1), get(1), prs(1), sccsfile(4).

DIAGNOSTICS
Use help(1) for explanations.
BUGS

`comb` may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.
NAME

conv — common object file converter

SYNOPSIS

conv [-a] [-o] [-p] -t target [- | files]

DESCRIPTION

The *conv* command converts object files in the common object file format from their current byte ordering to the byte ordering of the *target* machine. The converted file is written to *file.v*. The *conv* command can be used on either the source (sending) or target (receiving) machine.

Command line options are:

- indicates that the names of *files* should be read from the standard input.

-\(a\) If the input file is an archive, produce the output file in the UNIX System V Release 2.0 portable archive format.

-\(o\) If the input file is an archive, produce the output file in the old (pre- UNIX System V) archive format.

-\(p\) If the input file is an archive, produce the output file in the UNIX System V Release 1.0 random access archive format.

-\(t\) *target* Convert the object file to the byte ordering of the machine (*target*) to which the object file is being shipped. This may be another host or a target machine. Legal values for *target* are: pdp, vax, ibm, x86, b16, n3b, mc68 and m32.

The *conv* command is meant to ease the problems created by a multi-host cross-compilation development environment. The *conv* command is best used within a procedure for shipping object files from one machine to another.

The *conv* command will recognize and produce archive files in three formats: the pre- UNIX System V format, the UNIX System V Release 1.0 random access format, and the UNIX System V Release 2.0 portable ASCII format. By default, *conv* will create the output archive file in the same format as the input file. To produce an output file in a different format than the input file, use the -\(a\), -\(o\), or -\(p\) option. If the output archive format is the same as the input format, the archive symbol table will be converted, otherwise the symbol table will be stripped from the archive. The *ar*(1) command with its -\(t\) and -\(s\) options must be used on the target machine to recreate the archive symbol table.

EXAMPLE

To ship object files from a VAX to a 3B2 Computer, execute the following commands:

```
conv -t m32 *.out
uucp *.out.v my3b2!~/rje/
```
DIAGNOSTICS

The diagnostics are self-explanatory. Fatal diagnostics on the command lines cause termination. Fatal diagnostics on an input file cause the program to continue to the next input file.

CAVEATS

The conv command will not convert archives from one format to another if both the source and target machines have the same byte ordering. The UNIX system tool convert(1) should be used for this purpose.

SEE ALSO

ar(1), convert(1), ar(4), a.out(4).
NAME
convert — convert archive files to common formats

SYNOPSIS
convert infile outfile

DESCRIPTION
The convert command transforms input infile to output outfile. Infile must be a UNIX System V Release 1.0 archive file and outfile will be the equivalent UNIX System V Release 2.0 archive file. All other types of input to the convert command will be passed unmodified from the input file to the output file (along with appropriate warning messages).

Infile must be different from outfile.

FILES
TMPDIR/conv* temporary files

TMPDIR is usually /usr/tmp but can be redefined by setting the environment variable TMPDIR [see tempnam() in tmpnam(3S)].

SEE ALSO
ar(1), tmpnam(3S), a.out(4), ar(4)
NAME
cpp — the C language preprocessor

SYNOPSIS

LIBDIR/cpp [ option ... ] [ ifile [ ofile ] ]

DESCRIPTION

The C language preprocessor, cpp, is invoked as the first pass of any C compilation by the cc(1) command. Thus cpp’s output is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, cpp and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of cpp other than through the cc(1) command is not suggested, since the functionality of cpp may someday be moved elsewhere. See m4(1) for a general macro processor.

cpp optionally accepts two file names as arguments. Ifile and ofile are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following options to cpp are recognized:

-P Preprocess the input without producing the line control information used by the next pass of the C compiler.

-C By default, cpp strips C-style comments. If the -C option is specified, all comments (except those found on cpp directive lines) are passed along.

-Uname

Remove any initial definition of name, where name is a reserved symbol that is predefined by the particular preprocessor. Following is the current list of these possibly reserved symbols. On AT&T 3B2 and 3B5 Computers, unix and one of u3b2 or u3b5 are defined.

| operating system:       | unix, dmert, gcos, ibm, os, tss |
| hardware:               | interdata, pdp11, u370, u3b, u3b5, u3b2, u3b20d, vax |
| UNIX system variant:    | RES, RT |
| lint(1):                | lint |

-Dname
-Dname=def

Define name with value def as if by a #define. If no =def is given, name is defined with value 1. The -D option has lower precedence than the -U option. That is, if the same name is used in both a -U option and a -D option, the name will be undefined regardless of the order of the options.

-T The -T option forces cpp to use only the first eight characters to distinguish preprocessor symbols and is included for backward compatibility.

-Idir Change the algorithm for searching for #include files whose names do not begin with / to look in dir before looking in the directories on the standard list. Thus, #include files whose names are enclosed in "" will be searched for first in the directory of the file with the #include line,
then in directories named in -I options, and last in directories on a standard list. For `#include` files whose names are enclosed in `<>`, the directory of the file with the `#include` line is not searched.

- **-Ydir** Use directory `dir` in place of the standard list of directories when searching for `#include` files.

- **-H** Print, one per line on standard error, the path names of included files.

Two special names are understood by `cpp`. The name `__LINE__` is defined as the current line number (as a decimal integer) as known by `cpp`, and `__FILE__` is defined as the current file name (as a C string) as known by `cpp`. They can be used anywhere (including in macros) just as any other defined name.

All `cpp` directive lines start with `#` in column 1. Any number of blanks and tabs is allowed between the `#` and the directive. The directives are:

- **`#define name token-string`**
  Replace subsequent instances of `name` with `token-string`.

- **`#define name( arg, ..., arg ) token-string`**
  Notice that there can be no space between `name` and the `. Replace subsequent instances of `name` followed by a (, a list of comma-separated sets of tokens, and a ) followed by `token-string`, where each occurrence of an `arg` in the `token-string` is replaced by the corresponding set of tokens in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded `token-string` unchanged. After the entire `token-string` has been expanded, `cpp` re-starts its scan for names to expand at the beginning of the newly created `token-string`.

- **`#undef name`**
  Cause the definition of `name` (if any) to be forgotten from now on. No additional tokens are permitted on the directive line after `name`.

- **`#ident "string"`**
  Put `string` into the .comment section of an object file.

- **`#include "filename"`**
  Include at this point the contents of `filename` (which will then be run through `cpp`). When the `<filename>` notation is used, `filename` is only searched for in the standard places. See the `-I` and `-Y` options above for more detail. No additional tokens are permitted on the directive line after the final " or >.

- **`#line integer-constant "filename"`**
  Causes `cpp` to generate line control information for the next pass of the C compiler. `Integer-constant` is the line number of the next line and `filename` is the file from which it comes. If "filename" is not given, the current file name is unchanged. No additional tokens are permitted on the directive line after the optional `filename`.

- **`#endif`**
  Ends a section of lines begun by a test directive (`#if`, `#ifdef`, or `#ifndef`). Each test directive must have a matching `#endif`. No additional tokens are permitted on the directive line.
### #ifdef name
The lines following will appear in the output if and only if *name* has been the subject of a previous `#define` without being the subject of an intervening `#undef`. No additional tokens are permitted on the directive line after *name*.

### #ifndef name
The lines following will appear in the output if and only if *name* has not been the subject of a previous `#define`. No additional tokens are permitted on the directive line after *name*.

### #if constant-expression
Lines following will appear in the output if and only if the `constant-expression` evaluates to non-zero. All binary non-assignment C operators, the `?:` operator, the unary `-`, `!`, and `-` operators are all legal in `constant-expression`. The precedence of the operators is the same as defined by the C language. There is also a unary operator `defined`, which can be used in `constant-expression` in these two forms: `defined ( name )` or `defined name`. This allows the utility of `#ifdef` and `#ifndef` in a `#if` directive. Only these operators, integer constants, and names which are known by `cpp` should be used in `constant-expression`. In particular, the `sizeof` operator is not available.

To test whether either of two symbols, `foo` and `fum`, are defined, use

```c
#if defined(foo) | defined(fum)
```

### #elif constant-expression
An arbitrary number of `#elif` directives is allowed between a `#if`, `#ifdef`, or `#ifndef` directive and a `#else` or `#endif` directive. The lines following the `#elif` directive will appear in the output if and only if the preceding test directive evaluates to zero, all intervening `#elif` directives evaluate to zero, and the `constant-expression` evaluates to non-zero. If `constant-expression` evaluates to non-zero, all succeeding `#elif` and `#else` directives will be ignored. Any `constant-expression` allowed in a `#if` directive is allowed in a `#elif` directive.

### #else
The lines following will appear in the output if and only if the preceding test directive evaluates to zero, and all intervening `#elif` directives evaluate to zero. No additional tokens are permitted on the directive line.

The test directives and the possible `#else` directives can be nested.

### FILES
- `INCDIR`: standard directory list for `#include` files, usually `/usr/include`
- `LIBDIR`: usually `/lib`

### SEE ALSO
- `cc(1)`, `lint(1)`, `m4(1)`.

27
DIAGNOSTICS
The error messages produced by `cpp` are intended to be self-explanatory. The line number and file name where the error occurred are printed along with the diagnostic.

NOTES
The unsupported `-W` option enables the `#class` directive. If it encounters a `#class` directive, `cpp` will exit with code 27 after finishing all other processing. This option provides support for "C with classes".

Because the standard directory for included files may be different in different environments, this form of `#include` directive:

```
#include <file.h>
```

should be used, rather than one with an absolute path, like:

```
#include "/usr/include/file.h"
```

cpp warns about the use of the absolute pathname.
NAME
cprs – compress a common object file

SYNOPSIS
cprs [-p] file1 file2

DESCRIPTION
The cprs command reduces the size of a common object file, file1, by removing
duplicate structure and union descriptors. The reduced file, file2, is produced as
output.
The sole option to cprs is:
-p  Print statistical messages including: total number of tags, total duplicate
tags, and total reduction of file1.

SEE ALSO
strip(1), a.out(4), syms(4).
CTRACE(1) (Advanced C Utilities) CTRACE(1)

NAME
ctrace — C program debugger

SYNOPSIS
ctrace [options] [file]

DESCRIPTION
The ctrace command allows you to follow the execution of a C program, statement-by-statement. The effect is similar to executing a shell procedure with the -x option. ctrace reads the C program in file (or from standard input if you do not specify file), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of ctrace into a temporary file because the cc(1) command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the bfs(1) or tail(1) commands.

The options commonly used are:
- f functions  Trace only these functions.
- v functions  Trace all but these functions.

You may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. You can request that variables be printed in additional formats, if appropriate, with these options:
- o Octal
- x Hexadecimal
- u Unsigned
- e Floating point

These options are used only in special circumstances:
- l n Check n consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.
- s Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.
- t n Trace n variables per statement instead of the default of 10 (the maximum number is 20). The Diagnostics section explains when to use this option.
- P Run the C preprocessor on the input before tracing it. You can also use the -D, -I, and -U cpp(1) options.
These options are used to tailor the run-time trace package when the traced program will run in a non-UNIX System environment:

-b Use only basic functions in the trace code, that is, those in \texttt{ctype(3C)}, \texttt{printf(3S)}, and \texttt{string(3C)}. These are usually available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have \texttt{signal(2)}, \texttt{fflush(3S)}, \texttt{longjmp(3C)}, or \texttt{setjmp(3C)}.

-p \texttt{string}

Change the trace print function from the default of 'printf'. For example, 'fprintf(stderr,' would send the trace to the standard error output.

-r f Use file \texttt{f} in place of the \texttt{runtime.c} trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the \texttt{-p} option).

\textbf{EXAMPLE}

If the file \texttt{lc.c} contains this C program:

\begin{verbatim}
1 #include <stdio.h>
2 main()  /* count lines in input */
3 {
4     int c, nl;
5
6     nl = 0;
7     while ((c = getchar()) != EOF)
8         if (c == \textbackslash n')
9             ++nl;
10     printf("%d\n", nl);
11 }
\end{verbatim}

and you enter these commands and test data:

\begin{verbatim}
cc lc.c
a.out
1
\texttt{cntl-d}
\end{verbatim}

the program will be compiled and executed. The output of the program will be the number \texttt{2}, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke \texttt{ctrace} with these commands:

\begin{verbatim}
ctrace lc.c >temp.c
cc temp.c
a.out
\end{verbatim}

the output will be:

\begin{verbatim}
2 main()
6     nl = 0;
/* nl == 0 */
7     while ((c = getchar()) != EOF)
\end{verbatim}

The program is now waiting for input. If you enter the same test data as before, the output will be:

\begin{verbatim}
/* c == 49 or '1' */
\end{verbatim}
8     if (c == '\n')
     /* c == 10 or \n */
9     ++nl;
     /* nl == 1 */
7     while ((c = getchar()) != EOF)
     /* c == 10 or \n */
8     if (c == '\n')
     /* c == 10 or \n */
9     ++nl;
     /* nl == 2 */
7     while ((c = getchar()) != EOF)

If you now enter an end of file character (cntl-d) the final output will be:
     /* c == -1 */
10    printf("%d\n", nl);
     /* nl == 2 */
    return

Note that the program output printed at the end of the trace line for the nl variable. Also note the return comment added by ctrace at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable c is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this if statement, you will probably realize that you used the assignment operator (=) in place of the equality operator (==). You can easily miss this error during code reading.

EXECUTION-TIME TRACE CONTROL
The default operation for ctrace is to trace the entire program file, unless you use the -f or -v options to trace specific functions. This does not give you statement-by-statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program.

You can do both of these by adding ctroff() and ctroff() function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with if statements, and you can even conditionally include this code because ctrace defines the CTRACE preprocessor variable. For example:

```
#ifdef CTRACE
    if (c == '!' && i > 1000)
        ctroff();
#endif
```

You can also call these functions from sdb(1) if you compile with the -g option. For example, to trace all but lines 7 to 10 in the main function, enter:

```
sdb a.out
main:7b ctroff()
main:11b ctroff()
```

You can also turn the trace off and on by setting static variable tr_ct_to 0 and 1,
respectively. This is useful if you are using a debugger that cannot call these functions directly.

**DIAGNOSTICS**

This section contains diagnostic messages from both `ctrace` and `cc(1)`, since the traced code often gets some `cc` warning messages. You can get `cc` error messages in some rare cases, all of which can be avoided.

**ctrace Diagnostics**

- **warning: some variables are not traced in this statement**
  Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the `-t` option to increase this number.

- **warning: statement too long to trace**
  This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

- **cannot handle preprocessor code, use -P option**
  This is usually caused by #ifdef/#endif preprocessor statements in the middle of a C statement, or by a semicolon at the end of a #define preprocessor statement.

- **'if ... else if' sequence too long**
  Split the sequence by removing an `else` from the middle.

- **possible syntax error, try -P option**
  Use the `-P` option to preprocess the `ctrace` input, along with any appropriate `-D`, `-I`, and `-U` preprocessor options. If you still get the error message, check the Warnings section below.

**Cc Diagnostics**

- **warning: illegal combination of pointer and integer**

- **warning: statement not reached**

- **warning: sizeof returns 0**
  Ignore these messages.

- **compiler takes size of function**
  See the `ctrace"possible syntax error"` message above.

- **yacc stack overflow**
  See the `ctrace"if ... else if"sequence too long"` message above.

- **out of tree space; simplify expression**
  Use the `-t` option to reduce the number of traced variables per statement from the default of 10. Ignore the "ctrace: too many variables to trace" warnings you will now get.

- **redeclaration of signal**
  Either correct this declaration of `signal(2)`, or remove it and `#include <signal.h>`.

**SEE ALSO**

- `signal(2)`, `ctype(3C)`, `fclose(3S)`, `printf(3S)`, `setjmp(3C)`, `string(3C)`.
- `bfs(1)`, `tail(1)` in the *User's Reference Manual*. 

33
WARNINGS
You will get a ctrace syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

ctrace assumes that BADMAG is a preprocessor macro, and that EOF and NULL are #defined constants. Declaring any of these to be variables, e.g., "int EOF;", will cause a syntax error.

BUGS
ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. ctrace may choose to print the address of an aggregate or use the wrong format (e.g., 3.149050e-311 for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

FILES
/usr/lib/ctrace/runtime.c run-time trace package
NAME
cxref – generate C program cross-reference

SYNOPSIS
cxref [ options ] files

DESCRIPTION
The cxref command analyzes a collection of C files and attempts to build a
 cross-reference table. cxref uses a special version of cpp to include #define’d
 information in its symbol table. It produces a listing on standard output of all
 symbols (auto, static, and global) in each file separately, or, with the -c option,
 in combination. Each symbol contains an asterisk (*) before the declaring refer­
 ence.

In addition to the –D, –I and –U options [which are interpreted just as they are
 by cc(1) and cpp(1)], the following options are interpreted by cxref:

–c Print a combined cross-reference of all input files.
–w <num>
   Width option which formats output no wider than <num> (decimal)
   columns. This option will default to 80 if <num> is not specified or is
   less than 51.
–o file Direct output to file.
–s Operate silently; do not print input file names.
–t Format listing for 80-column width.

FILES
   LLIBDIR usually /usr/lib
   LLIBDIR/xcpp special version of the C preprocessor.

SEE ALSO
   cc(1), cpp(1).

DIAGNOSTICS
Error messages are unusually cryptic, but usually mean that you cannot compile
these files.

BUGS
   cxref considers a formal argument in a #define macro definition to be a declara­
   tion of that symbol. For example, a program that #includes ctype.h, will contain
   many declarations of the variable c.
NAME
delta – make a delta (change) to an SCCS file

SYNOPSIS
delta [-rSID] [-s] [-n] [-glist] [-m[mrlist]] [-y[comment]] [-p] files

DESCRIPTION
delta is used to permanently introduce into the named SCCS file changes that
were made to the file retrieved by get(1) (called the g-file, or generated file).
delta makes a delta to each named SCCS file. If a directory is named, delta
behaves as though each file in the directory were specified as a named file,
except that non-SCCS files (last component of the path name does not begin
with s.) and unreadable files are silently ignored. If a name of - is given, the
standard input is read (see WARNINGS); each line of the standard input is taken
to be the name of an SCCS file to be processed.
delta may issue prompts on the standard output depending upon certain
keyletters specified and flags [see admin(1)] that may be present in the SCCS file
(see -m and -y keyletters below).
Keyletter arguments apply independently to each named file.

-rSID  Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or
more outstanding gets for editing (get -e) on the same
SCCS file were done by the same person (login name). The SID value specified with the -r keyletter can be either
the SID specified on the get command line or the SID to be
made as reported by the get command [see get(1)]. A
diagnostic results if the specified SID is ambiguous, or, if
necessary and omitted on the command line.

-s  Suppresses the issue, on the standard output, of the
created delta's SID, as well as the number of lines inserted,
deleted and unchanged in the SCCS file.

-n  Specifies retention of the edited g-file (normally removed
at completion of delta processing).

-glist  a list (see get(1) for the definition of list) of deltas which
are to be ignored when the file is accessed at the change
level (SID) created by this delta.

-m[mrlist]  If the SCCS file has the v flag set [see admin(1)] then a
Modification Request (MR) number must be supplied as the
reason for creating the new delta.
If -m is not used and the standard input is a terminal, the
prompt MRs? is issued on the standard output before the
standard input is read; if the standard input is not a ter­
ninal, no prompt is issued. The MRs? prompt always pre­
cedes the comments? prompt (see -y keyletter).
MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the v flag has a value [see admin(1)], it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, delta terminates. (It is assumed that the MR numbers were not all valid.)

-y[comment]  Arbitrary text used to describe the reason for making the delta. A null string is considered a valid comment.

If -y is not specified and the standard input is a terminal, the prompt comments? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

-p  Causes delta to print (on the standard output) the SCCS file differences before and after the delta is applied in a diff(1) format.

FILES

g-file  Existed before the execution of delta; removed after completion of delta.
p-file  Existed before the execution of delta; may exist after completion of delta.
q-file  Created during the execution of delta; removed after completion of delta.
x-file  Created during the execution of delta; renamed to SCCS file after completion of delta.
z-file  Created during the execution of delta; removed during the execution of delta.
d-file  Created during the execution of delta; removed after completion of delta.
/usr/bin/bdiff  Program to compute differences between the "gotten" file and the g-file.

WARNINGS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS [see sccsfile(4) (5)] and will cause an error.

A get of many SCCS files, followed by a delta of those files, should be avoided when the get generates a large amount of data. Instead, multiple get/delta sequences should be used.

If the standard input (-) is specified on the delta command line, the -m (if necessary) and -y keyletters must also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.
SEE ALSO
  admin(1), cdc(1), get(1), prs(1), rmdel(1), sccsfile(4).

DIAGNOSTICS
  Use help(1) for explanations.
NAME
dis – object code disassembler

SYNOPSIS

DESCRIPTION
The dis command produces an assembly language listing of file, which may be an object file or an archive of object files. The listing includes assembly statements and an octal or hexadecimal representation of the binary that produced those statements.

The following options are interpreted by the disassembler and may be specified in any order.

-o
Print numbers in octal. The default is hexadecimal.

-V
Print, on standard error, the version number of the disassembler being executed.

-L
Lookup source labels in the symbol table for subsequent printing. This option works only if the file was compiled with additional debugging information [e.g., the -g option of cc(1)].

-s
Perform symbolic disassembly - i.e., specify source symbol names for operands where possible. Symbolic disassembly output will appear on the line following the instruction. For maximal symbolic disassembly to be performed, the file must be compiled with additional debugging information [e.g., the -g option of cc(1)]. Symbol names will be printed using C syntax.

-d sec
Disassemble the named section as data, printing the offset of the data from the beginning of the section.

-da sec
Disassemble the named section as data, printing the actual address of the data.

-F function
Disassemble only the named function in each object file specified on the command line. The -F option may be specified multiple times on the command line.

-t sec
Disassemble the named section as text.

-l string
Disassemble the library file specified by string. For example, one would issue the command dis -l x -l z to disassemble libx.a and libz.a. All libraries are assumed to be in LIBDIR.

If the -d, -da or -t options are specified, only those named sections from each user-supplied file name will be disassembled. Otherwise, all sections containing text will be disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as [5], represents that the break-pointable line number starts with the following instruction. These line numbers will be printed only if the file was compiled with additional debugging information [e.g., the -g option of cc(1)]. An expression such as <40> in the operand field or in the symbolic disassembly, following a relative displacement for control transfer instructions, is the computed address.
within the section to which control will be transferred. A function name will appear in the first column, followed by ()

**FILES**

**LIBDIR** usually /lib.

**SEE ALSO**

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

**DIAGNOSTICS**

The self-explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.
NAME
dump — dump selected parts of an object file

SYNOPSIS
dump [ options ] files

DESCRIPTION
The dump command dumps selected parts of each of its object file arguments. This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following options:
- Dump the archive header of each member of each archive file argument.
- Dump the global symbols in the symbol table of an archive.
- Dump each file header.
- Dump each optional header.
- Dump section headers.
- Dump section contents.
- Dump relocation information.
- Dump line number information.
- Dump symbol table entries.
- Dump line number entries for the named function.
- Dump the string table.
- Interpret and print the contents of the .lib sections.

The following modifiers are used in conjunction with the options listed above to modify their capabilities.
- Dump the section number, number, or the range of sections starting at number and ending at the number specified by +d.
+ Dump sections in the range either beginning with first section or beginning with section specified by -d.
- Dump information pertaining only to the named entity. This modifier applies to -h, -s, -r, -l, and -t.
- Suppress printing of the headers.
- Dump only the indexed symbol table entry. The -t used in conjunction with +t, specifies a range of symbol table entries.
+ Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the -t option.
- Underline the name of the file for emphasis.
-v    Dump information in symbolic representation rather than numeric (e.g., C_STATIC instead of 0X02). This modifier can be used with all the above options except -s and -o options of dump.

-z name,number    Dump line number entry or range of line numbers starting at number for the named function.

+z number    Dump line numbers starting at either function name or number specified by -z, up to number specified by +z.

Blanks separating an option and its modifier are optional. The comma separating the name from the number modifying the -z option may be replaced by a blank.

The dump command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

SEE ALSO
a.out(4), ar(4).
NAME

gencc – create a front-end to the cc command

SYNOPSIS

gencc

DESCRIPTION

The gencc command is an interactive command designed to aid in the creation of
a front-end to the cc command. Since hard-coded pathnames have been elimi­
nated from the C Compilation System (CCS), it is possible to move pieces of
the CCS to new locations without recompiling the CCS. The new locations of
moved pieces can be specified through the -Y option to the cc command. How­
ever, it is inconvenient to supply the proper -Y options with every invocation of
the cc command. Further, if a system administrator moves pieces of the CCS,
such movement should be invisible to users.

The front-end to the cc command which gencc generates is a one-line shell script
which calls the cc command with the proper -Y options specified. The front­
end to the cc command will also pass all user supplied options to the cc com­
mand.

gencc prompts for the location of each tool and directory which can be
respecified by a -Y option to the cc command. If no location is specified, it
assumes that that piece of the CCS has not been relocated. After all the loca­
tions have been prompted for, gencc will create the front-end to the cc com­
mand.

gencc creates the front-end to the cc command in the current working directory
and gives the file the same name as the cc command. Thus, gencc can not be
run in the same directory containing the actual cc command. Further, if a system
administrator has redistributed the CCS, the actual cc command should be
placed somewhere which is not typically in a user’s PATH (e.g., /lib). This will
prevent users from accidentally invoking the cc command without using the
front-end.

CAVEATS

gencc does not produce any warnings if a tool or directory does not exist at the
specified location. Also, gencc does not actually move any files to new locations.

FILES

./cc         front-end to cc

SEE ALSO

cc(1).
NAME
get – get a version of an SCCS file
SYNOPSIS
DESCRIPTION
get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with -. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, get behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the g-file whose name is derived from the SCCS file name by simply removing the leading s.; (see also FILES, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

-rSID
The SCCS IDentification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 (pg. 47) shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by delta(1) if the -e keyletter is also used), as a function of the SID specified.

-ccutoff
Cutoff date-time, in the form:

YY[MM][DD][HH][MM][SS][lllll]

No changes (deltas) to the SCCS file which were created after the specified cutoff date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, -c7502 is equivalent to -c750228235959. Any number of non-numeric characters may separate the various 2-digit pieces of the cutoff date-time. This feature allows one to specify a cutoff date in the form: "-c77/2/2 9:22:25". Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested gets within, say the input to a send(1C) command:

"!get "-c%E% %U%" s.file

-ilist
A list of deltas to be included (forced to be applied) in the creation of the generated file. The list has the following syntax:

<l|> ::= <range> | <l|> , <range>
<range> ::= SID | SID – SID
SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1.

-xlist A list of deltas to be excluded in the creation of the generated file. See the -i keyletter for the list format.

-e Indicates that the get is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of delta(1). The -e keyletter used in a get for a particular version (SID) of the SCCS file prevents further gets for editing on the same SID until delta is executed or the j (joint edit) flag is set in the SCCS file [see admin(1)]. Concurrent use of get -e for different SIDs is always allowed.

If the g-file generated by get with an -e keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the get command with the -k keyletter in place of the -e keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file [see admin(1)] are enforced when the -e keyletter is used.

-b Used with the -e keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the b flag is not present in the file [see admin(1)] or if the retrieved delta is not a leaf delta. (A leaf delta is one that has no successors on the SCCS file tree.) Note: A branch delta may always be created from a non-leaf delta. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

-k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The -k keyletter is implied by the -e keyletter.

-l[p] Causes a delta summary to be written into an l-file. If -lp is used then an l-file is not created; the delta summary is written on the standard output instead. See FILES for the format of the l-file.

-p Causes the text retrieved from the SCCS file to be written on the standard output. No g-file is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the -s keyletter is used, in which case it disappears.

-s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.
-m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

-n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the -m and -n keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the -m keyletter generated format.

-g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an l-file, or to verify the existence of a particular SID.

-t Used to access the most recently created delta in a given release (e.g., -r1), or release and level (e.g., -r1.2).

-w string Substitute string for all occurrences of %W% when getting the file.

-aseq-no. The delta sequence number of the SCCS file delta (version) to be retrieved [see scsfile(5)]. This keyletter is used by the comb(1) command; it is not a generally useful keyletter. If both the -r and -a keyletters are specified, only the -a keyletter is used. Care should be taken when using the -a keyletter in conjunction with the -e keyletter, as the SID of the delta to be created may not be what one expects. The -r keyletter can be used with the -a and -e keyletters to control the naming of the SID of the delta to be created.

For each file processed, get responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the -e keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the -i keyletter is used included deltas are listed following the notation “Included”; if the -x keyletter is used, excluded deltas are listed following the notation “Excluded”.
**TABLE 1. Determination of SCCS Identification String**

<table>
<thead>
<tr>
<th>SID* Specified</th>
<th>–b Keyletter Used†</th>
<th>Other Conditions Retrieved to be Created</th>
<th>SID of Delta Retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>none‡</td>
<td>no</td>
<td>R defaults to mR</td>
<td>mR.mL</td>
</tr>
<tr>
<td>none‡</td>
<td>yes</td>
<td>R defaults to mR</td>
<td>mR.mL</td>
</tr>
<tr>
<td>R</td>
<td>no</td>
<td>R &gt; mR</td>
<td>mR.mL</td>
</tr>
<tr>
<td>R</td>
<td>yes</td>
<td>R = mR</td>
<td>mR.mL</td>
</tr>
<tr>
<td>R</td>
<td>–</td>
<td>R ≤ mR and R does not exist</td>
<td>hR.mL**</td>
</tr>
<tr>
<td>R</td>
<td>–</td>
<td>Trunk succ.# in release &gt; R and R exists</td>
<td>R.mL</td>
</tr>
<tr>
<td>R.L</td>
<td>no</td>
<td>No trunk succ.</td>
<td>R.L</td>
</tr>
<tr>
<td>R.L</td>
<td>yes</td>
<td>No trunk succ.</td>
<td>R.L</td>
</tr>
<tr>
<td>R.L</td>
<td>–</td>
<td>Trunk succ. in release ≥ R</td>
<td>R.L</td>
</tr>
<tr>
<td>R.L.B</td>
<td>no</td>
<td>No branch succ.</td>
<td>R.L.B.mS</td>
</tr>
<tr>
<td>R.L.B</td>
<td>yes</td>
<td>No branch succ.</td>
<td>R.L.B.mS</td>
</tr>
<tr>
<td>R.L.B.S</td>
<td>no</td>
<td>No branch succ.</td>
<td>R.L.B.S</td>
</tr>
<tr>
<td>R.L.B.S</td>
<td>yes</td>
<td>No branch succ.</td>
<td>R.L.B.S</td>
</tr>
<tr>
<td>R.L.B.S</td>
<td>–</td>
<td>Branch succ.</td>
<td>R.L.B.S</td>
</tr>
</tbody>
</table>

* **R**, **L**, **B**, and **S** are the “release”, “level”, “branch”, and “sequence” components of the SID, respectively; “m” means maximum. Thus, for example, “R.mL” means “the maximum level number within release R’”; “R.(mB+1).L” means “the first sequence number on the new branch (i.e., maximum branch number plus one) of level L within release R’”. Note that if the SID specified is of the form “R.L”, “R.L.B”, or “R.L.B.S”, each of the specified components must exist.

** hR’’ is the highest existing release that is lower than the specified, nonexistent, release R.

*** This is used to force creation of the first delta in a new release.

# Successor.

† The –b keyletter is effective only if the b flag [see admin(1)] is present in the file. An entry of – means “irrelevant”.

‡ This case applies if the d (default SID) flag is not present in the file. If the d flag is present in the file, then the SID obtained from the d flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

**IDENTIFICATION KEYWORDS**

Identifying information is inserted into the text retrieved from the SCCS file by replacing identification keywords with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:
Keyword | Value
---|---
%M% | Module name: either the value of the m flag in the file [see admin(1)], or if absent, the name of the SCCS file with the leading s. removed.
%M% | SCCS identification (SID) (%R%.%L%-%B%.%S%) of the retrieved text.
%R% | Release.
%L% | Level.
%B% | Branch.
%S% | Sequence.
%D% | Current date (YY/MM/DD).
%H% | Current date (MM/DD/YY).
%T% | Current time (HH:MM:SS).
%E% | Date newest applied delta was created (YY/MM/DD).
%G% | Date newest applied delta was created (MM/DD/YY).
%U% | Time newest applied delta was created (HH:MM:SS).
%Y% | Module type: value of the t flag in the SCCS file [see admin(1)].
%F% | SCCS file name.
%P% | Fully qualified SCCS file name.
%Q% | The value of the q flag in the file [see admin(1)].
%C% | Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is not intended to be used on every line to provide sequence numbers.
%Z% | The 4-character string @(#) recognizable by what(1).
%W% | A shorthand notation for constructing what(1) strings for UNIX system program files. %W% = %Z%-%M%<horizontal-tab>%I%
%A% | Another shorthand notation for constructing what(1) strings for non-UNIX system program files.
%A% = %Z%-%Y%-%M%-%I%-%Z%

Several auxiliary files may be created by get. These files are known generically as the g-file, l-file, p-file, and z-file. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form s.module-name, the auxiliary files are named by replacing the leading s with the tag. The g-file is an exception to this scheme: the g-file is named by removing the s. prefix. For example, s.xyz.c, the auxiliary file names would be xyz.c, l.xyz.c, p.xyz.c, and z.xyz.c, respectively.

The g-file, which contains the generated text, is created in the current directory (unless the -p keyletter is used). A g-file is created in all cases, whether or not any lines of text were generated by the get. It is owned by the real user. If the -k keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The l-file contains a table showing which deltas were applied in generating the retrieved text. The l-file is created in the current directory if the -l keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the l-file have the following format:
a. A blank character if the delta was applied;  
   * otherwise.

b. A blank character if the delta was applied or was not applied  
   and ignored;  
   * if the delta was not applied and was not ignored.

c. A code indicating a "special" reason why the delta was or was  
   not applied:  
   "I": Included.  
   "X": Excluded.  
   "C": Cut off (by a -c keyletter).

d. Blank.

e. SCCS identification (SID).

f. Tab character.

g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.

h. Blank.

i. Login name of person who created \textit{delta}.

The comments and MR data follow on subsequent lines, indented one  
horizontal tab character. A blank line terminates each entry.

The \textit{p-file} is used to pass information resulting from a \textit{get}  
with an \texttt{-e} keyletter along to \textit{delta}. Its contents are also used  
to prevent a subsequent execution of \textit{get} with an \texttt{-e} keyletter  
for the same SID until \textit{delta} is executed or the joint edit  
flag, j, [see \textit{admin(1)}] is set in the SCCS file. The \textit{p-file} is  
created in the directory containing the SCCS file and the effective user  
must have write permission in that directory. Its mode is 644 and it is  
owned by the effective user. The format of the \textit{p-file} is: the gotten  
SID, followed by a blank, followed by the SID that the new delta will  
have when it is made, followed by a blank, followed by the login  
name of the real user, followed by a blank, followed by the date-time the  
\textit{get} was executed, followed by a blank and the \texttt{-i} keyletter argument  
if it was present, followed by a blank and the \texttt{-x} keyletter argument if it  
was present, followed by a new-line. There can be an arbitrary number of  
lines in the \textit{p-file} at any time; no two lines can have the same new delta SID.

The \textit{z-file} serves as a lock-out mechanism against simultaneous updates. Its  
contents are the binary (2 bytes) process ID of the command (i.e., \textit{get})  
that created it. The \textit{z-file} is created in the directory containing the  
SCCS file for the duration of \textit{get}. The same protection restrictions as those for the \textit{p-file} apply for the \textit{z-file}.  
The \textit{z-file} is created mode 444.

\begin{tabular}{ll}
\textbf{FILES} & \\
\textbf{g-file} & Existed before the execution of \textit{delta}; removed after completion  
   of \textit{delta}.
\textbf{p-file} & Existed before the execution of \textit{delta}; may exist after completion  
   of \textit{delta}.
\textbf{q-file} & Created during the execution of \textit{delta}; removed after completion  
   of \textit{delta}.
\textbf{x-file} & Created during the execution of \textit{delta}; renamed to SCCS file  
   after completion of \textit{delta}.
\textbf{z-file} & Created during the execution of \textit{delta}; removed during the execution  
   of \textit{delta}.
\end{tabular}
d-file  Created during the execution of `delta`; removed after completion of `delta`.

`/usr/bin/bdiff`  Program to compute differences between the "gotten" file and the `g-file`.

SEE ALSO
  `admin(1), delta(1), prs(1), what(1)`.

DIAGNOSTICS
  Use `help(1)` for explanations.

BUGS
  If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the `-e` keyletter is used.
NAME
infocmp — compare or print out terminfo descriptions

SYNOPSIS
[-w width] [-A directory] [-B directory] [termname ...]

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

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

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

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

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

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

Source Listing Options [-I] [-L] [-C] [-r]
The -I, -L, and -C options will produce a source listing for each terminal
named.

-I use the terminfo(4) names

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

-C use the termcap names

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

If no termnames are given, the environment variable TERM will be used for the
terminal name.
The source produced by the `-C` option may be used directly as a `termcap` entry, but not all of the parameterized strings may be changed to the `termcap` format. `infocmp` will attempt to convert most of the parameterized information, but that which it doesn’t will be plainly marked in the output and commented out. These should be edited by hand.

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

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

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

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

<table>
<thead>
<tr>
<th>Terminfo</th>
<th>Termcap</th>
<th>Representative Terminals</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%p1%c</code></td>
<td><code>%.</code></td>
<td>adm</td>
</tr>
<tr>
<td><code>%p1%d</code></td>
<td><code>%d</code></td>
<td>hp, ANSI standard, vt100</td>
</tr>
<tr>
<td><code>%p1%'x'%%%c</code></td>
<td><code>%+%c</code></td>
<td>concept</td>
</tr>
<tr>
<td><code>%i</code></td>
<td><code>%i</code></td>
<td>ANSI standard, vt100</td>
</tr>
<tr>
<td><code>%p1%?'x%'&gt;'y'%%%;</code></td>
<td><code>&gt;%xy</code></td>
<td>concept</td>
</tr>
<tr>
<td><code>%p2 is printed before %p1</code></td>
<td><code>%r</code></td>
<td>hp</td>
</tr>
</tbody>
</table>

**Use═ Option [−u]**

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

A capability will get printed with an at-sign (@) if it no longer exists in the first `termname`, but one of the other `termname` entries contains a value for it. A capability’s value gets printed if the value in the first `termname` is not found in any of the other `termname` entries, or if the first of the other `termname` entries that has this capability gives a different value for the capability than that in the first `termname`.
The order of the other \textit{termname} entries is significant. Since the terminfo compiler \texttt{tic(1M)} does a left-to-right scan of the capabilities, specifying two \texttt{use=} entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given in. \texttt{infocmp} will flag any such inconsistencies between the other \textit{termname} entries as they are found.

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

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

\textbf{Other Options} \([-s\ \texttt{dillic}]\texttt{[\[-v\]}\texttt{[\[-V\]}\texttt{[\[-1\}]}\texttt{[\[-w\ width]}\]
\begin{itemize}
  \item \texttt{-s} \texttt{} sort the fields within each type according to the argument below:
    \begin{itemize}
      \item \texttt{d} leave fields in the order that they are stored in the \texttt{terminfo} database.
      \item \texttt{i} sort by \texttt{terminfo} name.
      \item \texttt{l} sort by the long \texttt{C} variable name.
      \item \texttt{c} sort by the \texttt{termcap} name.
    \end{itemize}
  \item If no \texttt{-s} option is given, the fields printed out will be sorted alphabetically by the \texttt{terminfo} name within each type, except in the case of the \texttt{-C} or the \texttt{-L} options, which cause the sorting to be done by the \texttt{termcap} name or the long \texttt{C} variable name, respectively.
  \item \texttt{-v} print out tracing information on standard error as the program runs.
  \item \texttt{-V} print out the version of the program in use on standard error and exit.
  \item \texttt{-1} cause the fields to printed out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.
  \item \texttt{-w} change the output to width characters.
\end{itemize}

\textbf{Changing Databases} \([-\texttt{A} \texttt{directory}]\texttt{[\[-\texttt{B} \texttt{directory}]\}

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

\textbf{FILES}\n\begin{itemize}
  \item \texttt{/usr/lib/terminfo/?/*} compiled terminal description database
\end{itemize}
INFOCMP(1M)       (Terminal Information Utilities)       INFOCMP(1M)

DIAGNOSTICS

malloc is out of space!

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

use= order dependency found:

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

‘use=term’ did not add anything to the description.

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

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

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

SEE ALSO

Chapter 10 of the Programmer's Guide.

NOTE

The termcap database (from earlier releases of UNIX System V) may not be supplied in future releases.
NAME
install — install commands

SYNOPSIS

DESCRIPTION
The install command is most commonly used in “makefiles” [See make(1)] to install a file (updated target file) in a specific place within a file system. Each file is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (dirx ...) are given, install will search a set of default directories (/bin, /usr/bin, /etc, /lib, and /usr/lib, in that order) for a file with the same name as file. When the first occurrence is found, install issues a message saying that it is overwriting that file with file, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (dirx ...) are specified after file, those directories will be searched before the directories specified in the default list.

The meanings of the options are:

- **-c dira** Installs a new command (file) in the directory specified by dira, only if it is not found. If it is found, install issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the -s option.

- **-f dirb** Forces file to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to 755 and bin, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the -o or -s options.

- **-i** Ignores default directory list, searching only through the given directories (dirx ...). May be used alone or with any other options except -c and -f.

- **-n dirc** If file is not found in any of the searched directories, it is put in the directory specified in dirc. The mode and owner of the new file will be set to 755 and bin, respectively. May be used alone or with any other options except -c and -f.

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

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

- **-g group** The group id of the new file is set to group. Only available to the superuser.
-o  If *file* is found, this option saves the "found" file by copying it to *OLDfile* in the directory in which it was found. This option is useful when installing a frequently used file such as */bin/sh* or */etc/getty*, where the existing file cannot be removed. May be used alone or with any other options except -c.

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

**SEE ALSO**
make(1)
NAME
ld — link editor for common object files

SYNOPSIS
ld [options] filename

DESCRIPTION
The ld command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and ld combines the objects, producing an object module that can either be executed or, if the -r option is specified, used as input for a subsequent ld run. The output of ld is left in a.out. By default this file is executable if no errors occurred during the load. If any input file, filename, is not an object file, ld assumes it is either an archive library or a text file containing link editor directives. [See Link Editor Directives in the UNIX System V Programmer's Guide for a discussion of input directives.]

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. The library may be either a relocatable archive library or a shared library. [See Shared Libraries in the UNIX System V Programmer's Guide for a discussion of shared libraries.] Only those routines defining an unresolved external reference are loaded. The library (archive) symbol table [see ar(4)] is searched sequentially with as many passes as are necessary to resolve external references which can be satisfied by library members. Thus, the ordering of library members is functionally unimportant, unless there exist multiple library members defining the same external symbol.

The following options are recognized by ld:

-e epsym
Set the default entry point address for the output file to be that of the symbol epsym.

-f fill
Set the default fill pattern for "holes" within an output section as well as initialized bss sections. The argument fill is a two-byte constant.

-Lx
Search a library libx.a, where x is up to nine characters. A library is searched when its name is encountered, so the placement of a -l is significant. By default, libraries are located in LIBDIR or LLIBDIR.

-m
Produce a map or listing of the input/output sections on the standard output.

-o outfile
Produce an output object file by the name outfile. The name of the default object file is a.out.

-r
Retain relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent ld run. The link editor will not complain about unresolved references, and the output file will not be executable.

-a
Create an absolute file. This is the default if the -r option is not used. Used with the -r option, -a allocates memory for common symbols.
-s  Strip line number entries and symbol table information from the output object file.

-t  Turn off the warning about multiply-defined symbols that are not the same size.

-u symname
Enter symname as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. The placement of this option on the ld line is significant; it must be placed before the library which will define the symbol.

-x  Do not preserve local symbols in the output symbol table; enter external and static symbols only. This option saves some space in the output file.

-z  Do not bind anything to address zero. This option will allow runtime detection of null pointers.

-L dir  Change the algorithm of searching for libx.a to look in dir before looking in LIBDIR and LLIBDIR. This option is effective only if it precedes the -I option on the command line.

-M  Output a message for each multiply-defined external definition.

-N  Put the text section at the beginning of the text segment rather than after all header information, and put the data section immediately following text in the core image.

-V  Output a message giving information about the version of ld being used.

-VS num
Use num as a decimal version stamp identifying the a.out file that is produced. The version stamp is stored in the optional header.

-Y[LU],dir  Change the default directory used for finding libraries. If L is specified the first default directory which ld searches, LIBDIR, is replaced by dir. If U is specified and ld has been built with a second default directory, LLIBDIR, then that directory is replaced by dir. If ld was built with only one default directory and U is specified a warning is printed and the option is ignored.

FILES
LIBDIR/libx.a libraries
LLIBDIR/libx.a libraries
a.out output file
LIBDIR usually /lib
LLIBDIR usually /usr/lib

SEE ALSO
as(1), cc(1), mkshlib(1), exit(2), end(3C), a.out(4), ar(4), and Link Editor Directives and Shared Libraries in the Programmer's Guide.
CAVEATS

Through its options and input directives, the common link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives and options should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the program’s address space.

- When the link editor is called through `cc(1)`, a startup routine is linked with the user’s program. This routine calls `exit()` [see `exit(2)`] after execution of the main program. If the user calls the link editor directly, then the user must insure that the program always calls `exit()` rather than falling through the end of the entry routine.

The symbols `etext`, `edata`, and `end` [see `end(3C)`] are reserved and are defined by the link editor. It is incorrect for a user program to redefine them.

If the link editor does not recognize an input file as an object file or an archive file, it will assume that it contains link editor directives and will attempt to parse it. This will occasionally produce an error message complaining about "syntax errors".

Arithmetic expressions may only have one forward referenced symbol per expression.
NAME
lex — generate programs for simple lexical tasks

SYNOPSIS
lex [ -rctvn ] [ file ] ...

DESCRIPTION
The lex command generates programs to be used in simple lexical analysis of
text.

The input files (standard input default) contain strings and expressions to be
searched for, and C text to be executed when strings are found.

A file lex.yy.c is generated which, when loaded with the library, copies the
input to the output except when a string specified in the file is found; then the
corresponding program text is executed. The actual string matched is left in
yytext, an external character array. Matching is done in order of the strings in
the file. The strings may contain square brackets to indicate character classes, as
in [abx-z] to indicate a, b, x, y, and z; and the operators *, +, and ? mean
respectively any non-negative number of, any positive number of, and either
zero or one occurrence of, the previous character or character class. The char­
cacter . is the class of all ASCII characters except new-line. Parentheses for
grouping and vertical bar for alternation are also supported. The notation r{d,e}
in a rule indicates between d and e instances of regular expression r. It has
higher precedence than \ but lower than *, ?, +, and concatenation. Thus
[a-zA-Z]+ matches a string of letters. The character ^ at the beginning of an
expression permits a successful match only immediately after a new-line, and
the character $ at the end of an expression requires a trailing new-line. The
character / in an expression indicates trailing context; only the part of the
expression up to the slash is returned in yytext, but the remainder of the expres­
sion must follow in the input stream. An operator character may be used as an
ordinary symbol if it is within " symbols or preceded by \.

Three subroutines defined as macros are expected: input() to read a character;
unput(c) to replace a character read; and output(c) to place an output character.
They are defined in terms of the standard streams, but you can override them.
The program generated is named yylex(), and the library contains a main() which calls it. The action REJECT on the right side of the rule causes this match
to be rejected and the next suitable match executed; the function yymore() accu­
mulates additional characters into the same yytext; and the function yyless(p)
pushes back the portion of the string matched beginning at p, which should be
between yytext and yytext+yylen. The macros input and output use files yyin
and yyo ut to read from and write to, defaulted to stdin and stdout, respectively.

Any line beginning with a blank is assumed to contain only C text and is
copied; if it precedes %% it is copied into the external definition area of the
lex.yy.c file. All rules should follow a %%, as in YACC. Lines preceding %%
which begin with a non-blank character define the string on the left to be the
remainder of the line; it can be called out later by surrounding it with { }. Note
that curly brackets do not imply parentheses; only string substitution is done.
EXAMPLE

D [0-9]
% if printf("IF statement\n");
[a-z]+ printf("tag, value %s\n", yytext);
0[D]+ printf("octal number %s\n", yytext);
[D]+ printf("decimal number %s\n", yytext);
"++" printf("unary op\n");
"++" printf("binary op\n");
"/*" skipcommnts();
%
skipcommnts()
{
    for (; ;)
    {
        while (inputO != '.
        if (inputO != ')
            unput(yytext[yylen-1]);
        else
            return;
    }
}

The external names generated by lex all begin with the prefix yy or YY.
The flags must appear before any files. The flag -r indicates RATFOR actions,
-c indicates C actions and is the default, -t causes the lex.yy.c program to be
written instead to standard output, -v provides a one-line summary of statistics,
-n will not print out the -v summary. Multiple files are treated as a single file.
If no files are specified, standard input is used.
Certain table sizes for the resulting finite state machine can be set in the
definitions section:

%p n number of positions is n (default 2500)
%n n number of states is n (500)
%e n number of parse tree nodes is n (1000)
%a n number of transitions is n (2000)
%k n number of packed character classes is n (1000)
%o n size of output array is n (3000)

The use of one or more of the above automatically implies the -v option, unless
the -n option is used.

SEE ALSO
yacc(1).
Programmer's Guide.

BUGS
The -r option is not yet fully operational.
NAME
lint – a C program checker

SYNOPSIS
lint [ option ] ... file ...

DESCRIPTION
The lint command attempts to detect features of the C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

Arguments whose names end with .c are taken to be C source files. Arguments whose names end with .ln are taken to be the result of an earlier invocation of lint with either the --c or the --o option used. The .ln files are analogous to .o (object) files that are produced by the cc(1) command when given a .c file as input. Files with other suffixes are warned about and ignored.

lint will take all the .c, .ln, and llib-lx.ln (specified by --lx) files and process them in their command line order. By default, lint appends the standard C lint library (llib-lc.ln) to the end of the list of files. However, if the --p option is used, the portable C lint library (llib-port.ln) is appended instead. When the --c option is not used, the second pass of lint checks this list of files for mutual compatibility. When the --c option is used, the .ln and the llib-lx.ln files are ignored.

Any number of lint options may be used, in any order, intermixed with filename arguments. The following options are used to suppress certain kinds of complaints:

- a Suppress complaints about assignments of long values to variables that are not long.
- b Suppress complaints about break statements that cannot be reached. (Programs produced by lex or yacc will often result in many such complaints).
- h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.
- u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running lint on a subset of files of a larger program).
- v Suppress complaints about unused arguments in functions.
- x Do not report variables referred to by external declarations but never used.
The following arguments alter lint’s behavior:

- `Ix` Include additional lint library `llib-llx.ln`. For example, you can include a lint version of the math library `llib-lm.ln` by inserting `-lm` on the command line. This argument does not suppress the default use of `llib-lc.ln`. These lint libraries must be in the assumed directory. This option can be used to reference local lint libraries and is useful in the development of multi-file projects.

- `n` Do not check compatibility against either the standard or the portable lint library.

- `p` Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

- `c` Cause lint to produce a `.In` file for every `.c` file on the command line. These `.In` files are the product of lint’s first pass only, and are not checked for inter-function compatibility.

- `o lib` Cause lint to create a lint library with the name `llibllib.ln`. The `-c` option nullifies any use of the `-o` option. The lint library produced is the input that is given to lint’s second pass. The `-o` option simply causes this file to be saved in the named lint library. To produce a `llibllib.ln` without extraneous messages, use of the `-x` option is suggested. The `-v` option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file `llib-lc` is written). These option settings are also available through the use of “lint comments” (see below).

The `-D`, `-U`, and `-I` options of `cpp(1)` and the `-g` and `-O` options of `cc(1)` are also recognized as separate arguments. The `-g` and `-O` options are ignored, but, by recognizing these options, lint’s behavior is closer to that of the `cc(1)` command. Other options are warned about and ignored. The pre-processor symbol “lint” is defined to allow certain questionable code to be altered or removed for lint. Therefore, the symbol “lint” should be thought of as a reserved word for all code that is planned to be checked by lint.

Certain conventional comments in the C source will change the behavior of lint:

```
/*NOTREACHED*/
  at appropriate points stops comments about unreachable code.
  [This comment is typically placed just after calls to functions like exit(2)].

/*VARARGSn*/
  suppresses the usual checking for variable numbers of arguments
  in the following function declaration. The data types of the first
  n arguments are checked; a missing n is taken to be 0.

/*ARGSUSED*/
  turns on the -v option for the next function.
```
/*LINTLIBRARY*/

at the beginning of a file shuts off complaints about unused
functions and function arguments in this file. This is equivalent
to using the -v and -x options.

`lint` produces its first output on a per-source-file basis. Complaints regarding
included files are collected and printed after all source files have been processed.
Finally, if the -c option is not used, information gathered from all input files is
collected and checked for consistency. At this point, if it is not clear whether a
complaint stems from a given source file or from one of its included files, the
source file name will be printed followed by a question mark.

The behavior of the -c and the -o options allows for incremental use of `lint` on
a set of C source files. Generally, one invokes `lint` once for each source file with
the -c option. Each of these invocations produces a .In file which corresponds
to the .c file, and prints all messages that are about just that source file. After all
the source files have been separately run through `lint`, it is invoked once more
(without the -c option), listing all the .In files with the needed -Ix options.
This will print all the inter-file inconsistencies. This scheme works well with
`make(1)`; it allows `make` to be used to `lint` only the source files that have been
modified since the last time the set of source files were linted.

FILES

`LLIBDIR` the directory where the lint libraries specified by the -Ix
option must exist, usually /usr/lib
`LLIBDIR/lint[12]` first and second passes
`LLIBDIR/llib-lc.ln` declarations for C Library functions (binary format; source
is in LLIBDIR/llib-lc)
`LLIBDIR/llib-port.ln` declarations for portable functions (binary format; source
is in LLIBDIR/llib-port)
`LLIBDIR/llib-lm.ln` declarations for Math Library functions (binary format;
source is in LLIBDIR/llib-lm)
`TMPDIR/*lint*` temporaries
`TMPDIR` usually /usr/tmp but can be redefined by setting the
environment variable TMPDIR [see `tempnam()` in
tmpnam(3S)].

SEE ALSO
cc(1), cpp(1), make(1).

BUGS
`exit(2), setjmp(3C)` and other functions that do not return are not understood;
this causes various lies.
NAME
list — produce C source listing from a common object file

SYNOPSIS
list [-V] [-h] [-F function] source-file . . . [object-file]

DESCRIPTION
The list command produces a C source listing with line number information
attached. If multiple C source files were used to create the object file, list will
accept multiple file names. The object file is taken to be the last non-C source
file argument. If no object file is specified, the default object file, a.out, will be
used.

Line numbers will be printed for each line marked as breakpoint inserted by the
compiler (generally, each executable C statement that begins a new line of
source). Line numbering begins anew for each function. Line number 1 is
always the line containing the left curly brace (lbrace) that begins the function body.
Line numbers will also be supplied for inner block redeclarations of local vari­
ables so that they can be distinguished by the symbolic debugger.

The following options are interpreted by list and may be given in any order:

- V          Print, on standard error, the version number of the list command
               executing.
- h          Suppress heading output.
- F function List only the named function. The - F option may be specified
               multiple times on the command line.

SEE ALSO
as(1), cc(1), ld(1).

CAVEATS
Object files given to list must have been compiled with the -g option of cc(1).
Since list does not use the C preprocessor, it may be unable to recognize func­
tion definitions whose syntax has been distorted by the use of C preprocessor
macro substitutions.

DIAGNOSTICS
list will produce the error message "list: name: cannot open" if name cannot be
read. If the source file names do not end in .c, the message is "list: name:
invalid C source name". An invalid object file will cause the message "list:
name: bad magic" to be produced. If some or all of the symbolic debugging
information is missing, one of the following messages will be printed: "list:
name: symbols have been stripped, cannot proceed", "list: name: cannot read
line numbers", and "list: name: not in symbol table". The following messages
are produced when list has become confused by #ifdef’s in the source file: "list:
name: cannot find function in symbol table", "list: name: out of sync; too many
)"", and "list: name: unexpected end-of-file". The error message "list: name:
missing or inappropriate line numbers" means that either symbol debugging
information is missing, or list has been confused by C preprocessor statements.
NAME
lorder — find ordering relation for an object library

SYNOPSIS
lorder file ...

DESCRIPTION
The input is one or more object or library archive files [see ar(1)]. The standard output is a list of pairs of object file or archive member names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by tsort(1) to find an ordering of a library suitable for one-pass access by ld(1). Note that the link editor ld(1) is capable of multiple passes over an archive in the portable archive format [see ar(4)] and does not require that lorder(1) be used when building an archive. The usage of the lorder(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing .o files.

```
ar -cr library `lorder *.o | tsort`
```

FILES

- `TMPDIR/*symref` temporary files
- `TMPDIR/*symdef` temporary files

`TMPDIR` is usually /usr/tmp but can be redefined by setting the environment variable TMPDIR [see tempnam() in tmpnam(3S)].

SEE ALSO
ar(1), ld(1), tsort(1), ar(4).

CAVEAT
lorder will accept as input any object or archive file, regardless of its suffix, provided there is more than one input file. If there is but a single input file, its suffix must be .o.
NAME
m4 - macro processor

SYNOPSIS
m4 [ options ] [ files ]

DESCRIPTION
The m4 command is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is -, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:
-e  Operate interactively. Interrupts are ignored and the output is unbuffered.
-s  Enable line sync output for the C preprocessor (#line ...)
-Bn  Change the size of the push-back and argument collection buffers from the default of 4,096.
-Hn  Change the size of the symbol table hash array from the default of 199. The size should be prime.
-Sn  Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.
-Tn  Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any -D or -U flags:
-Dname [=val]
  Defines name to val or to null in val's absence.
-Uname
  undefines name.

Macro calls have the form:
  name(arg1,arg2,...,argn)

The ( must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.
m4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

**define**

the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $n$ in the replacement text, where $n$ is a digit, is replaced by the $n$-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; $#$ is replaced by the number of arguments; $*$ is replaced by a list of all the arguments separated by commas; $@$ is like $*$, but each argument is quoted (with the current quotes).

**undefine**

removes the definition of the macro named in its argument.

**defn**

returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

**pushdef**

like define, but saves any previous definition.

**popdef**

removes current definition of its argument(s), exposing the previous one, if any.

**ifdef**

if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word unix is predefined on UNIX system versions of m4.

**shift**

returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

**changequote**

change quote symbols to the first and second arguments. The symbols may be up to five characters long. Changequote without arguments restores the original values (i.e., `').

**changecom**

change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.

**divert**

m4 maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The divert macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

**undivert**

causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

**divnum**

returns the value of the current output stream.

**dnl**

reads and discards characters up to and including the next newline.
ifelse has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

decr returns the value of its argument decremented by 1.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponentiation), bitwise & and |; relational; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.

substr returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include returns the contents of the file named in the argument.

sinclude is identical to include, except that it says nothing if the file is inaccessible.

syscmd executes the UNIX system command given in the first argument. No value is returned.

sysval is the return code from the last call to syscmd.

maketemp fills in a string of XXXXX in its argument with the current process ID.

m4exit causes immediate exit from m4. Argument 1, if given, is the exit code; the default is 0.

m4wrap argument 1 will be pushed back at final EOF; example: m4wrap("cleanup()")

erprint prints its argument on the diagnostic output file.

dumpdef prints current names and definitions, for the named items, or for all if no arguments are given.
traceon with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.

traceoff turns off trace globally and for any macros specified. Macros specifically traced by traceon can be untraced only by specific calls to traceoff.

SEE ALSO
cc(1), cpp(1).

*The m4 Macro Processor* in the *Support Tools Guide.*
NAME
make — maintain, update, and regenerate groups of programs

SYNOPSIS

DESCRIPTION
The make command allows the programmer to maintain, update, and regenerate groups of computer programs. The following is a brief description of all options and some special names:

- **-f makefile** Description file name. makefile is assumed to be the name of a description file.
- **-p** Print out the complete set of macro definitions and target descriptions.
- **-i** Ignore error codes returned by invoked commands. This mode is entered if the fake target name .IGNORE appears in the description file.
- **-k** Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.
- **-s** Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name .SILENT appears in the description file.
- **-r** Do not use the built-in rules.
- **-n** No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.
- **-b** Compatibility mode for old makefiles.
- **-e** Environment variables override assignments within makefiles.
- **-u** Force an unconditional update.
- **-t** Touch the target files (causing them to be up-to-date) rather than issue the usual commands.
- **-q** Question. The make command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

.DEFAULT If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name .DEFAULT are used if it exists.

.PRECIOUS Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT Same effect as the -s option.

.IGNORE Same effect as the -i option.

make executes commands in makefile to update one or more target names. Name is typically a program. If no -f option is present, makefile, Makefile, and the Source Code Control System(SCCS) files s.makefile, and s.Makefile are tried in
order. If *makefile* is --, the standard input is taken. More than one -- *makefile* argument pair may appear.

make updates a target only if its dependents are newer than the target (unless the --u option is used to force an unconditional update). All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date.

*makefile* contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a :, then a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that begin with a tab are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash> <new-line> sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```plaintext
  echo a\b
```

will produce

```plaintext
  ab
```

exactly the same as the shell would.

Sharp (#) and new-line surround comments.

The following *makefile* says that *pgm* depends on two files *a.o* and *b.o*, and that they in turn depend on their corresponding source files (*a.c* and *b.c*) and a common file *incl.h*:

```plaintext
pgm: a.o b.o
    cc a.o b.o -o pgm
a.o: incl.h a.c
    cc -c a.c
b.o: incl.h b.c
    cc -c b.c
```

Command lines are executed one at a time, each by its own shell. The SHELL environment variable can be used to specify which shell make should use to execute commands. The default is /bin/sh. The first one or two characters in a command can be the following: --, --, or --. If @ is present, printing of the command is suppressed. If -- is present, make ignores an error. A line is printed when it is executed unless the --s option is present, or the entry .SILENT: is in *makefile*, or unless the initial character sequence contains a @. The --n option specifies printing without execution; however, if the command line has the string $(MAKE) in it, the line is always executed (see discussion of the *MAKEFLAGS* macro under *Environment*). The --t (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate make. If the --i option is present, or the entry .IGNORE: appears in *makefile*, or the initial character sequence of the command contains --, the error is ignored. If the --k option is
present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The `-b` option allows old makefiles (those written for the old version of `make`) to run without errors.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name `.PRECIOUS`.

Environment
The environment is read by `make`. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The `-e` option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built-in rules.

The `MAKEFLAGS` environment variable is processed by `make` as containing any legal input option (except `-f` and `-p`) defined for the command line. Further, upon invocation, `make` “invents” the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, `MAKEFLAGS` always contains the current input options. This proves very useful for “super-makes”. In fact, as noted above, when the `-n` option is used, the command `$(MAKE)` is executed anyway; hence, one can perform a `make -n` recursively on a whole software system to see what would have been executed. This is because the `-n` is put in `MAKEFLAGS` and passed to further invocations of `$(MAKE)`. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Include Files
If the string `include` appears as the first seven letters of a line in a `makefile`, and is followed by a blank or a tab, the rest of the line is assumed to be a file name and will be read by the current invocation, after substituting for any macros.

Macros
Entries of the form `string1 = string2` are macro definitions. `String2` is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of `$(string1[;subst1=[subst2]])` are replaced by `string2`. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional `:subst1=[subst2]` is a substitute sequence. If it is specified, all non-overlapping occurrences of `subst1` in the named macro are replaced by `subst2`. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under Libraries.

Internal Macros
There are five internally maintained macros which are useful for writing rules for building targets.

`$*` The macro `$*` stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.
$@$ The $@$ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$<$ The $<$ macro is only evaluated for inference rules or the .DEFAULT rule. It is the module which is out-of-date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the .c.o rule, the $<$ macro would evaluate to the .c file. An example for making optimized .o files from .c files is:

```
c.c -c -O $*.c
```
or:

```
c.c -c -O $<
```

$? The $? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules which must be rebuilt.

$% The $% macro is only evaluated when the target is an archive library member of the form lib(file.o). In this case, $@ evaluates to lib and $% evaluates to the library member, file.o.

Four of the five macros can have alternative forms. When an upper case D or F is appended to any of the four macros, the meaning is changed to "directory part" for D and "file part" for F. Thus, $(@D) refers to the directory part of the string $@. If there is no directory part, .j is generated. The only macro excluded from this alternative form is $?.

Suffixes

Certain names (for instance, those ending with .o) have inferable prerequisites such as .c, .s, etc. If no update commands for such a file appear in makefile, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, make has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c* .f .f* .sh .sh*
.c.o .c.a .c-o .c-.c .c-.a
.f.o .f.a .f-.o .f-.f .f-.a
.h-.h .s.o .s-.o .s-.s .s-.a .sh-.sh
.l.o .l.c .l-.o .l-.l .l-.c
.y.o .y.c .y-.o .y-.y .y-.c
```

The internal rules for make are contained in the source file rules.c for the make program. These rules can be locally modified. To print out the rules compiled into the make on any machine in a form suitable for recompilation, the following command is used:

```
make -fp - 2> /dev/null < /dev/null
```

A tilde in the above rules refers to an SCCS file [see sccsfile(4)]. Thus, the rule .c-.o would transform an SCCS C source file into an object file (.o). Because the
s. of the SCCS files is a prefix, it is incompatible with `make's` suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., `.c`) is the definition of how to build `x` from `x.c`. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for `.SUFFIXES`. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

```
.SUFFIXES: .o .c .y .y- .l .l- .s .s- .sh .sh- .h .h- .f .f-
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; `.SUFFIXES: with no dependencies clears the list of suffixes.

**Inference Rules**

The first example can be done more briefly.

```
pgm: a.o b.o
   cc a.o b.o -o pgm
   a.o b.o: incl.h
```

This is because `make` has a set of internal rules for building files. The user may add rules to this list by simply putting them in the `makefile`.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, `CFLAGS`, `LFLAGS`, and `YFLAGS` are used for compiler options to `cc`, `lex`, and `yacc`, respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix `.o` from a file with suffix `.c` is specified as an entry with `.c.o:` as the target and no dependents. Shell commands associated with the target define the rule for making a `.o` file from a `.c` file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

**Libraries**

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus `lib(file.o)` and `$LIB(file.o)` both refer to an archive library which contains `file.o`. (This assumes the `LIB` macro has been previously defined.) The expression `$LIB(file1.o file2.o)` is not legal. Rules pertaining to archive libraries have the form `.XX.a` where the `XX` is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the `XX` to be different from the suffix of the archive member. Thus, one cannot have `lib(file.o)` depend upon `file.o` explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:   lib(file1.o) lib(file2.o) lib(file3.o)
      @echo lib is now up-to-date
```
.c.a:

```make
$(CC) -c $(CFLAGS) $<
$(AR) $(ARFLAGS) $@ $*.o
rm -f $*.o
```

In fact, the `.c.a` rule listed above is built into `make` and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
$(CC) -c $(CFLAGS) $(?:.o=.c)
$(AR) $(ARFLAGS) lib $? 
rm $? @echo lib is now up-to-date
```

Here the substitution mode of the macro expansions is used. The `?` list is defined to be the set of object file names (inside `lib`) whose C source files are out-of-date. The substitution mode translates the `.o` to `.c`. (Unfortunately, one cannot as yet transform to `.c`; however, this may become possible in the future.) Note also, the disabling of the `.c.a:` rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

**FILES**

`[Mm]akefile` and `s.[Mm]akefile`

`/bin/sh`

**SEE ALSO**

`cc(1), lex(1), yacc(1), printf(3S), sccsfile(4), cd(1), sh(1)` in the *User’s Reference Manual*.

**NOTES**

Some commands return non-zero status inappropriately; use `-i` to overcome the difficulty.

**BUGS**

File names with the characters `= : @` will not work. Commands that are directly executed by the shell, notably `cd(1)`, are ineffectual across new-lines in `make`. The syntax `(lib(file1.o file2.o file3.o)` is illegal. You cannot build `lib(file.o)` from `file.o`. The macro `$(a:o=.c)` does not work. Named pipes are not handled well.
NAME
mcs — manipulate the object file comment section

SYNOPSIS
mcs [options] object-file ...

DESCRIPTION
The mcs command manipulates the comment section, normally the ".comment" section, in an object file. It is used to add to, delete, print, and compress the contents of the comment section in a UNIX System object file. mcs must be given one or more of the options described below. It takes each of the options given and applies them in order to the object-files.

If the object file is an archive, the file is treated as a set of individual object files. For example, if the -a option is specified, the string is appended to the comment section of each archive element.

The following options are available.
-a string
Append string to the comment section of the object-files. If string contains embedded blanks, it must be enclosed in quotation marks.
-c
Compress the contents of the comment section. All duplicate entries are removed. The ordering of the remaining entries is not disturbed.
-d
Delete the contents of the comment section from the object file. The object file comment section header is removed also.
-n name
Specify the name of the section to access. By default, mcs deals with the section named .comment. This option can be used to specify another section.
-p
Print the contents of the comment section on the standard output. If more than one name is specified, each entry printed is tagged by the name of the file from which it was extracted, using the format "filename:string."

EXAMPLES
mcs -p file # Print file's comment section.
mcs -a string file # Append string to file's comment section

FILES
TMPDIR/mcs* temporary files
TMPDIR/* temporary files
TMPDIR is usually /usr/tmp but can be redefined by setting the environment variable TMPDIR [see tempnam() in tmpnam(3S)].

SEE ALSO
cpp(1), a.out(4).

NOTES
mcs cannot add new sections or delete existing sections to executable objects with magic number 0413 [see a.out(4)].
mkshlib — create a shared library

**SYNOPSIS**

```bash
mkshlib -s specfil [-t target] [-h host] [-n] [-q]
```

**DESCRIPTION**

The `mkshlib` command builds both the host and target shared libraries. A shared library is similar in function to a normal, non-shared library, except that programs which link with a shared library will share the library code during execution whereas programs which link with a non-shared library will get their own copy of each library routine used.

The host shared library is an archive which is used to link-edit user programs with the shared library [see `ar(4)`]. A host shared library can be treated exactly like a non-shared library and should be included on `cc(1)` command lines in the usual way [see `cc(1)`]. Further, all operations which can be performed on an archive can also be performed on the host shared library.

The target shared library is an executable module which is attached to the user’s process during execution of a program using the shared library. The target shared library contains the code for all the routines in the library and must be fully resolved. The target will be brought into memory during execution of a program using the shared library, and subsequent processes which use the shared library will share the copy of code already in memory. The text of the target is always shared, but each process will get its own copy of the data.

The user interface to `mkshlib` consists of command line options and a shared library specification file. The shared library specification file describes the contents of the shared library.

The `mkshlib` command invokes other tools such as the archiver, `ar(1)`, the assembler, `as(1)`, and the loader, `ld(1)`. Tools are invoked through the use of `system(3S)` which searches directories in the user’s PATH. Also, prefixes to `mkshlib` are parsed in the same manner as prefixes to the `cc(1)` command, and invoked tools are given the prefix, where appropriate. For example, `3b2mkshlib` will invoke `3b2ld`.

The following command line options are recognized by `mkshlib`:

- `-s specfil` Specifies the shared library specification file, `specfil`. This file contains the information necessary to build a shared library. Its contents include the branch table specifications for the target, the path-name in which the target should be installed, the start addresses of text and data for the target, the initialization specifications for the host, and the list of object files to be included in the shared library (see details below).

- `-t target` Specifies the name, `target`, of the target shared library produced on the host machine. When `target` is moved to the target machine, it should be installed at the location given in the specification file (see the `#target` directive below). If the `-n` option is used, then a new target shared library will not be generated.
MKSHLIB(1)  (Advanced C Utilities)  MKSHLIB(1)

-h host  Specifies the name of the host shared library, host. If this option is not given, then the host shared library will not be produced.

-n  Do not generate a new target shared library. This option is useful when producing only a new host shared library. The -t option must still be supplied since a version of the target shared library is needed to build the host shared library.

-q  Quiet warning messages. This option is useful when warning messages are expected but not desired.

The shared library specification file contains all the information necessary to build both the host and target shared libraries. The contents and format of the specification file are given by the following directives:

#address sectname address
   Specifies the start address, address, of section sectname for the target. This directive typically is used to specify the start addresses of the .text and .data sections.

#target pathname
   Specifies the absolute pathname, pathname, of the target shared library on the target machine. This pathname is copied to a.out files and is the location where the operating system will look for the shared library when executing a file which uses it.

#branch
   Specifies the start of the branch table specifications. The lines following this directive are taken to be branch table specification lines. Branch table specification lines have the following format:

   funcname <white space> position

   where funcname is the name of the symbol given a branch table entry and position specifies the position of funcname’s branch table entry. Position may be a single integer or a range of integers of the form position1-position2. Each position must be greater than or equal to one, the same position can not be specified more than once, and every position from one to the highest given position must be accounted for.

   If a symbol is given more than one branch table entry by associating a range of positions with the symbol or by specifying the same symbol on more than one branch table specification line, then the symbol is defined to have the address of the highest associated branch table entry. All other branch table entries for the symbol can be thought of as "empty" slots and can be replaced by new entries in future versions of the shared library.

   Finally, only functions should be given branch table entries, and those functions must be external.
This directive can be specified only once per shared library specification file.

#objects

Specifies the names of the object files constituting the target shared library. The lines following this directive are taken to be the list of input object files in the order they are to be loaded into the target. The list simply consists of each filename followed by white space. This list is also used to determine the input object files for the host shared library, but the order for the host is given by running the list through lorder(1) and tsort(1).

This directive can be specified only once per shared library specification file.

#init object

Specifies that the object file, object, requires initialization code. The lines following this directive are taken to be initialization specification lines.

Initialization specification lines have the following format:

    pimport <white space> import

Pimport is a pointer to the associated imported symbol, import, and must be defined in the current specified object file, object. The initialization code generated for each such line is of the form:

    pimport = &import;

where pimpadder is the absolute address of pimport.

All initializations for a particular object file must be given at once and multiple specifications of the same object file are not allowed.

#ident string

Specifies a string, string, to be included in the .comment section of the target shared library. This directive can be specified only once per shared library specification file.

##

Specifies a comment. All information on a line following this directive is ignored.

All directives which may be followed by multi-line specifications are valid until the next directive or the end of the file.

FILES

    TEMPDIR/* temporary files

TEMPDIR is usually /usr/tmp but can be redefined by setting the environment variable TMPDIR [see tempnam() in tempnam(3S)].
SEE ALSO
   ar(1), as(1), cc(1), ld(1), a.out(4), ar(4).
   Chapter 8 ("Shared Libraries") in the *Programmer's Guide*.
NAME

nm — print name list of common object file

SYNOPSIS

nm [−oxhvnefurpVT] filename ...

DESCRIPTION

The nm command displays the symbol table of each common object file, filename. Filename may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed:

Name The name of the symbol.
Value Its value expressed as an offset or an address depending on its storage class.
Class Its storage class.
Type Its type and derived type. If the symbol is an instance of a structure or of a union then the structure or union tag will be given following the type (e.g., struct-tag). If the symbol is an array, then the array dimensions will be given following the type (e.g., char[n][m]). Note that the object file must have been compiled with the −g option of the cc(1) command for this information to appear.
Size Its size in bytes, if available. Note that the object file must have been compiled with the −g option of the cc(1) command for this information to appear.
Line The source line number at which it is defined, if available. Note that the object file must have been compiled with the −g option of the cc(1) command for this information to appear.
Section For storage classes static and external, the object file section containing the symbol (e.g., text, data or bss).

The output of nm may be controlled using the following options:

−o Print the value and size of a symbol in octal instead of decimal.
−x Print the value and size of a symbol in hexadecimal instead of decimal.
−h Do not display the output header data.
−v Sort external symbols by value before they are printed.
−n Sort external symbols by name before they are printed.
−e Print only external and static symbols.
−f Produce full output. Print redundant symbols (.text, .data, .lib, and .bss), normally suppressed.
−u Print undefined symbols only.
−r Prepend the name of the object file or archive to each output line.
−p Produce easily parsable, terse output. Each symbol name is preceded by its value (blanks if undefined) and one of the letters U (undefined),
A (absolute), T (text segment symbol), D (data segment symbol), S (user defined segment symbol), R (register symbol), F (file symbol), or C (common symbol). If the symbol is local (non-external), the type letter is in lower case.

-V
Print the version of the nm command executing on the standard error output.

-T
By default, nm prints the entire name of the symbols listed. Since object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The -T option causes nm to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both nm name -e -v and nm -ve name print the static and external symbols in name, with external symbols sorted by value.

FILES
TMPDIR/* temporary files

TMPDIR is usually /usr/tmp but can be redefined by setting the environment variable TMPDIR [see tempnam() in tmpnam(3S)].

BUGS
When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the -v and -n options should be used only in conjunction with the -e option.

SEE ALSO
as(1), cc(1), ld(1), tmpnam(3S), a.out(4), ar(4).

DIAGNOSTICS
"nm: name: cannot open"
   if name cannot be read.

"nm: name: bad magic"
   if name is not a common object file.

"nm: name: no symbols"
   if the symbols have been stripped from name.
NAME
prof – display profile data

SYNOPSIS
prof [-tcan] [-ox] [-g] [-z] [-h] [-s] [-m mdata] [prog]

DESCRIPTION
The prof command interprets a profile file produced by the monitor(3C) function. The symbol table in the object file prog (a.out by default) is read and correlated with a profile file (mon.out by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options t, c, a, and n determine the type of sorting of the output lines:
- t  Sort by decreasing percentage of total time (default).
- c  Sort by decreasing number of calls.
- a  Sort by increasing symbol address.
- n  Sort lexically by symbol name.

The mutually exclusive options o and x specify the printing of the address of each symbol monitored:
- o  Print each symbol address (in octal) along with the symbol name.
- x  Print each symbol address (in hexadecimal) along with the symbol name.

The following options may be used in any combination:
- g  Include non-global symbols (static functions).
- z  Include all symbols in the profile range [see monitor(3C)], even if associated with zero number of calls and zero time.
- h  Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)
- s  Print a summary of several of the monitoring parameters and statistics on the standard error output.
- m mdata
   Use file mdata instead of mon.out as the input profile file.

A program creates a profile file if it has been loaded with the -p option of cc(1). This option to the cc command arranges for calls to monitor(3C) at the beginning and end of execution. It is the call to monitor at the end of execution that causes a profile file to be written. The number of calls to a function is tallied if the -p option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable PROFDIR. If PROFDIR does not exist, “mon.out” is produced in the directory that is current when the program terminates. If PROFDIR = string, “string/pid.progname” is produced, where progname consists of argv[0] with any
path prefix removed, and $pid$ is the program’s process id. If PROFDIR is the null string, no profiling output is produced.

A single function may be split into subfunctions for profiling by means of the MARK macro [see prof(5)].

FILES

mon.out for profile
a.out for namelist

SEE ALSO

cc(1), exit(2), profil(2), monitor(3C), prof(5).

WARNING

The times reported in successive identical runs may show variances of 20% or more, because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may “beat” with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely.

The times for static functions are attributed to the preceding external text symbol if the -g option is not used. However, the call counts for the preceding function are still correct, i.e., the static function call counts are not added in with the call counts of the external function.

CAVEATS

Only programs that call exit(2) or return from main will cause a profile file to be produced, unless a final call to monitor is explicitly coded.

The use of the -p option to cc(1) to invoke profiling imposes a limit of 600 functions that may have call counters established during program execution. For more counters you must call monitor(3C) directly. If this limit is exceeded, other data will be overwritten and the mon.out file will be corrupted. The number of call counters used will be reported automatically by the prof command whenever the number exceeds 5/6 of the maximum.
NAME
prs — print an SCCS file

SYNOPSIS
prs [-d[dataspec]] [-r[SID]] [-e] [-l] [-c[date-time]] [-a] files

DESCRIPTION
prs prints, on the standard output, parts or all of an SCCS file [see sccsfile(4)] in a user-supplied format. If a directory is named, prs behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.), and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to prs, which may appear in any order, consist of keyletter arguments, and file names.

All the described keyletter arguments apply independently to each named file:

- d[dataspec] Used to specify the output data specification. The dataspec is a string consisting of SCCS file data keywords (see DATA KEYWORDS) interspersed with optional user supplied text.
- r[SID] Used to specify the SCCS IDentification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.
- e Requests information for all deltas created earlier than and including the delta designated via the -r keyletter or the date given by the -c option.
- l Requests information for all deltas created later than and including the delta designated via the -r keyletter or the date given by the -c option.
- c date-time The cutoff date-time -c[cutoff]] is in the form:

YY[MM][DD][HH][MM][SS][Slllll]

c data-time The cutoff date-time -c[cutoff]] is in the form:

YY[MM][DD][HH][MM][SS][Slllll]

- c[date-time] Units omitted from the date-time default to their maximum possible values; that is, -c7502 is equivalent to -c750228235959. Any number of non-numeric characters may separate the various 2-digit pieces of the cutoff date in the form: "-c77/2/2 9:22:25".
- a Requests printing of information for both removed, i.e., delta type = R, [see rmdel(1)] and existing, i.e., delta type = D, deltas. If the -a keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS
Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file [see sccsfile(4)] have an associated data
keyword. There is no limit on the number of times a data keyword may appear in a dataspec.

The information printed by prs consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the dataspec. The format of a data keyword value is either Simple (S), in which keyword substitution is direct, or Multi-line (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords. A tab is specified by \t and carriage return/new-line is specified by \n. The default data keywords are:

":Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"

### TABLE 1. SCCS Files Data Keywords

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Data Item</th>
<th>File Section</th>
<th>Value</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Dt:</td>
<td>Delta information</td>
<td>Delta Table</td>
<td>See below*</td>
<td>S</td>
</tr>
<tr>
<td>:DL:</td>
<td>Delta line statistics</td>
<td></td>
<td>:Li:/:Ld:/:Lu:</td>
<td>S</td>
</tr>
<tr>
<td>:Li:</td>
<td>Lines inserted by Delta</td>
<td></td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:Ld:</td>
<td>Lines deleted by Delta</td>
<td></td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:Lu:</td>
<td>Lines unchanged by Delta</td>
<td></td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:DT:</td>
<td>Delta type</td>
<td></td>
<td>D 'or' R</td>
<td>S</td>
</tr>
<tr>
<td>:I:</td>
<td>SCCS ID string (SID)</td>
<td></td>
<td>:R::L::B::S:</td>
<td>S</td>
</tr>
<tr>
<td>:R:</td>
<td>Release number</td>
<td></td>
<td>nnnn</td>
<td>S</td>
</tr>
<tr>
<td>:L:</td>
<td>Level number</td>
<td></td>
<td>nnnn</td>
<td>S</td>
</tr>
<tr>
<td>:B:</td>
<td>Branch number</td>
<td></td>
<td>nnnn</td>
<td>S</td>
</tr>
<tr>
<td>:S:</td>
<td>Sequence number</td>
<td></td>
<td>nnnn</td>
<td>S</td>
</tr>
<tr>
<td>:D:</td>
<td>Date Delta created</td>
<td></td>
<td>:Dy:/:Dm:/:Dd:</td>
<td>S</td>
</tr>
<tr>
<td>:Dy:</td>
<td>Year Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Dm:</td>
<td>Month Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Dd:</td>
<td>Day Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:T:</td>
<td>Time Delta created</td>
<td></td>
<td>:Th:::Tm:::Ts:</td>
<td>S</td>
</tr>
<tr>
<td>:Th:</td>
<td>Hour Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Tm:</td>
<td>Minutes Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Ts:</td>
<td>Seconds Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:P:</td>
<td>Programmer who created Delta</td>
<td></td>
<td>logname</td>
<td>S</td>
</tr>
<tr>
<td>:DS:</td>
<td>Delta sequence number</td>
<td></td>
<td>nnnn</td>
<td>S</td>
</tr>
<tr>
<td>:DP:</td>
<td>Predecessor Delta seq-no.</td>
<td></td>
<td>nnnn</td>
<td>S</td>
</tr>
<tr>
<td>:DI:</td>
<td>Seq-no. of deltas incl., excl., ignored</td>
<td></td>
<td>:Dn:/:Dx:/:Dg:</td>
<td>S</td>
</tr>
<tr>
<td>:Dn:</td>
<td>Deltas included (seq #)</td>
<td></td>
<td>:DS:/:DS::</td>
<td>S</td>
</tr>
<tr>
<td>:Dx:</td>
<td>Deltas excluded (seq #)</td>
<td></td>
<td>:DS:/:DS::</td>
<td>S</td>
</tr>
<tr>
<td>:Dg:</td>
<td>Deltas ignored (seq #)</td>
<td></td>
<td>:DS:/:DS::</td>
<td>S</td>
</tr>
<tr>
<td>:MR:</td>
<td>MR numbers for delta</td>
<td></td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:C:</td>
<td>Comments for delta</td>
<td></td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:UN:</td>
<td>User names</td>
<td>User Names</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:FL:</td>
<td>Flag list</td>
<td>Flags</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:Y:</td>
<td>Module type flag</td>
<td></td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:MF:</td>
<td>MR validation flag</td>
<td></td>
<td>yes 'or' no</td>
<td>S</td>
</tr>
</tbody>
</table>
### TABLE 1. SCCS Files Data Keywords (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Data Item</th>
<th>File Section</th>
<th>Value</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MP:</td>
<td>MR validation pgm name</td>
<td>&quot;</td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:KF:</td>
<td>Keyword error/warning flag</td>
<td>&quot;</td>
<td>yes &quot;or&quot; no</td>
<td>S</td>
</tr>
<tr>
<td>:KV:</td>
<td>Keyword validation string</td>
<td>&quot;</td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:BF:</td>
<td>Branch flag</td>
<td>&quot;</td>
<td>yes &quot;or&quot; no</td>
<td>S</td>
</tr>
<tr>
<td>:J:</td>
<td>Joint edit flag</td>
<td>&quot;</td>
<td>yes &quot;or&quot; no</td>
<td>S</td>
</tr>
<tr>
<td>:LK:</td>
<td>Locked releases</td>
<td>&quot;</td>
<td>:R:....</td>
<td>S</td>
</tr>
<tr>
<td>:Q:</td>
<td>User-defined keyword</td>
<td>&quot;</td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:M:</td>
<td>Module name</td>
<td>&quot;</td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:FB:</td>
<td>Floor boundary</td>
<td>&quot;</td>
<td>:R:</td>
<td>S</td>
</tr>
<tr>
<td>:CB:</td>
<td>Ceiling boundary</td>
<td>&quot;</td>
<td>:R:</td>
<td>S</td>
</tr>
<tr>
<td>:DS:</td>
<td>Default SID</td>
<td>&quot;</td>
<td>:S:</td>
<td>S</td>
</tr>
<tr>
<td>:ND:</td>
<td>Null delta flag</td>
<td>&quot;</td>
<td>yes &quot;or&quot; no</td>
<td>S</td>
</tr>
<tr>
<td>:FD:</td>
<td>File descriptive text</td>
<td>Comments</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:BD:</td>
<td>Body</td>
<td>Body</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:GB:</td>
<td>Gotten body</td>
<td>&quot;</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:W:</td>
<td>A form of what(1) string</td>
<td>N/A</td>
<td>:Z::M:\t:i:</td>
<td>S</td>
</tr>
<tr>
<td>:A:</td>
<td>A form of what(1) string</td>
<td>N/A</td>
<td>:Z::Y::M::I::Z:</td>
<td>S</td>
</tr>
<tr>
<td>:Z:</td>
<td>what(1) string delimiter</td>
<td>N/A</td>
<td>@(#)</td>
<td>S</td>
</tr>
<tr>
<td>:F:</td>
<td>SCCS file name</td>
<td>N/A</td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:PN:</td>
<td>SCCS file path name</td>
<td>N/A</td>
<td>text</td>
<td>S</td>
</tr>
</tbody>
</table>

* :Dt:=""DT:":"D:"":T:"":P:"":DS:"":DP:*

**EXAMPLES**

prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

Users and/or user IDs for s.file are:

xyz
131
abc

prs -d"Newest delta for pgm :M:: :l: Created :D: By :P:" -r s.file

may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a special case:

prs s.file

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the special case is the -a keyletter.
FILES
    /tmp/pr????

SEE ALSO
    admin(1), delta(1), get(1), sccsfile(4),

DIAGNOSTICS
    Use help(1) for explanations.
NAME
   regcmp – regular expression compile

SYNOPSIS
   regcmp [ − ] files

DESCRIPTION
   The regcmp command performs a function similar to regcmp(3X) and, in most
cases, precludes the need for calling regcmp(3X) from C programs. This saves on
both execution time and program size. The command regcmp compiles the reg­
ular expressions in file and places the output in file.i. If the − option is used,
the output will be placed in file.c. The format of entries in file is a name (C
variable) followed by one or more blanks followed by a regular expression
enclosed in double quotes. The output of regcmp is C source code. Compiled
regular expressions are represented as extern char vectors. File.i files may thus
be included in C programs, or file.c files may be compiled and later loaded. In
the C program which uses the regcmp output, regex(abc,line) will apply the reg­
ular expression named abc to line. Diagnostics are self-explanatory.

EXAMPLES
   name "((A−Za−z][A−Za−z0−9_]*$0"
   telno "((0,1)((2−9][01][1−9]$0\(0,1) *"
      "((2−9][0−9][2])$1[ −][0,1]"
      "((0−9][4])$2"

In the C program that uses the regcmp output,
   regex(telno, line, area, exch, rest)
   will apply the regular expression named telno to line.

SEE ALSO
   regcmp(3X).
NAME
   relogin - rename login entry to show current layer

SYNOPSIS
   /usr/lib/layersys/relogin [-s] [line]

DESCRIPTION
   The relogin command changes the terminal line field of a user's utmp(4) entry to
   the name of the windowing terminal layer attached to standard input. write(1)
   messages sent to this user are directed to this layer. In addition, the who(1)
   command will show the user associated with this layer. relogin may only be
   invoked under layers(1).

   relogin is invoked automatically by layers(1) to set the utmp(4) entry to the ter-
   minal line of the first layer created upon startup, and to reset the utmp(4) entry
   to the real line on termination. It may be invoked by a user to designate a
   different layer to receive write(1) messages.

   -s    Suppress error messages.

   line  Specifies which utmp(4) entry to change. The utmp(4) file is searched
         for an entry with the specified line field. That field is changed to the
         line associated with the standard input. (To learn what lines are associ­
         ated with a given user, say jdoe, type ps -f -u jdoe and note the values
         shown in the TTY field (see ps(1))).

FILES
   /etc/utmp    database of users versus terminals

EXIT STATUS
   Returns 0 upon successful completion, 1 otherwise.

SEE ALSO

NOTES
   If line does not belong to the user issuing the relogin command or standard input
   is not associated with a terminal, relogin will fail.
NAME
rmdel — remove a delta from an SCCS file

SYNOPSIS
rmdel -rSID files

DESCRIPTION
rmdel removes the delta specified by the SID from each named SCCS file. The
delta to be removed must be the newest (most recent) delta in its branch in the
delta chain of each named SCCS file. In addition, the specified must not be that
of a version being edited for the purpose of making a delta (i.e., if a p-file [see
get(1)] exists for the named SCCS file, the specified must not appear in any entry
of the p-file).
The -r option is used for specifying the SID (SCCS IDentification) level of the
delta to be removed.
If a directory is named, rmdel behaves as though each file in the directory were
specified as a named file, except that non-SCCS files (last component of the path
name does not begin with s.) and unreadable files are silently ignored. If a
name of - is given, the standard input is read; each line of the standard input is
taken to be the name of an SCCS file to be processed; non-SCCS files and
unreadable files are silently ignored.
Simply stated, they are either (1) if you make a delta you can remove it; or (2) if
you own the file and directory you can remove a delta.

FILES
x.file [see delta(1)]
z.file [see delta(1)]

SEE ALSO
delta(1), get(1), prs(1), sccsfile(4).

DIAGNOSTICS
Use help(1) for explanations.
NAME
   sact — print current SCCS file editing activity

SYNOPSIS
   sact files

DESCRIPTION
   sact informs the user of any impending deltas to a named SCCS file. This situa-
   tion occurs when get(1) with the -e option has been previously executed
   without a subsequent execution of delta(1). If a directory is named on the com-
   mand line, sact behaves as though each file in the directory were specified as a
   named file, except that non-SCCS files and unreadable files are silently ignored.
   If a name of - is given, the standard input is read with each line being taken as
   the name of an SCCS file to be processed.

   The output for each named file consists of five fields separated by spaces.
   
   Field 1 specifies the SID of a delta that currently exists in the SCCS
   file to which changes will be made to make the new delta.
   
   Field 2 specifies the SID for the new delta to be created.
   
   Field 3 contains the logname of the user who will make the delta
   (i.e., executed a get for editing).
   
   Field 4 contains the date that get -e was executed.
   
   Field 5 contains the time that get -e was executed.

SEE ALSO
   delta(1), get(1), unget(1).

DIAGNOSTICS
   Use help(1) for explanations.
NAME
sccsdiff — compare two versions of an SCCS file

SYNOPSIS
sccsdiff -rSID1 -rSID2 [-p] [-sn] files

DESCRIPTION
sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

-\(rSID\)
  \(SID1\) and \(SID2\) specify the deltas of an SCCS file that are to be compared. Versions are passed to \(bdiff(1)\) in the order given.

-p
  pipe output for each file through \(pr(1)\).

-sn \(n\)
  \(n\) is the file segment size that \(bdiff\) will pass to \(diff(1)\). This is useful when \(diff\) fails due to a high system load.

FILES
/tmp/get????? Temporary files

SEE ALSO
get(1),

DIAGNOSTICS
"file: No differences" If the two versions are the same.
Use help(1) for explanations.
NAME
sdb - symbolic debugger

SYNOPSIS
sdb [-w] [-W] [objfil [corfil [directory-list]]]

DESCRIPTION
The sdb command calls a symbolic debugger that can be used with C and F77 programs. It may be used to examine their object files and core files and to provide a controlled environment for their execution.

Objfil is an executable program file which has been compiled with the -g (debug) option. If it has not been compiled with the -g option the symbolic capabilities of sdb will be limited, but the file can still be examined and the program debugged. The default for objfil is a.out. Corfil is assumed to be a core image file produced after executing objfil; the default for corfil is core. The core file need not be present. A - in place of corfil will force sdb to ignore any core image file. The colon separated list of directories (directory-list) is used to locate the source files used to build objfil.

It is useful to know that at any time there is a current line and current file. If corfil exists then they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in main(). The current line and file may be changed with the source file examination commands.

By default, warnings are provided if the source files used in producing objfil cannot be found, or are newer than objfil. This checking feature and the accompanying warnings may be disabled by the use of the -W flag.

Names of variables are written just as they are in C or F77. sdb does not truncate names. Variables local to a procedure may be accessed using the form procedure:variable. If no procedure name is given, the procedure containing the current line is used by default.

It is also possible to refer to structure members as variable.member, pointers to structure members as variable->member and array elements as variable[number]. Pointers may be dereferenced by using the form pointer[0]. Combinations of these forms may also be used. F77 common variables may be referenced by using the name of the common block instead of the structure name. Blank common variables may be named by the form .variable. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by sdb. An unqualified structure variable may also be used with various commands. Generally, sdb will interpret a structure as a set of variables. Thus, sdb will display the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value sdb displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as variable [number][number]... or as variable [number,number,...]. In place of number, the form number;number may be used to indicate a range of values, * may be used to
indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, sdb displays all the values of an array or of the section of an array if trailing subscripts are omitted. It displays only the address of the array itself or of the section specified by the user if subscripts are omitted. A multidimensional parameter in an F77 program cannot be displayed as an array, but it is actually a pointer, whose value is the location of the array. The array itself can be accessed symbolically from the calling function.

A particular instance of a variable on the stack may be referenced by using the form procedure:variable,number. All the variations mentioned in naming variables may be used. Number is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C may be used, so that addresses may be input in decimal, octal or hexadecimal.

Line numbers in the source program are referred to as file-name:number or procedure:number. In either case the number is relative to the beginning of the file. If no procedure or file name is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under sdb, all addresses refer to the executing program; otherwise they refer to objfil or corfil. An initial argument of -w permits overwriting locations in objfil.

Addresses

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (b1, e1, f1) and (b2, e2, f2) and the file address corresponding to a written address is calculated as follows:

\[
\begin{align*}
  b1 & \leq \text{address} < e1 \\
  \text{file address} & = \text{address} + f1 - b1 \\
  \text{otherwise} & \\
  b2 & \leq \text{address} < e2 \\
  \text{file address} & = \text{address} + f2 - b2,
\end{align*}
\]

otherwise, the requested address is not legal. In some cases (e.g., for programs with separated I and D space) the two segments for a file may overlap.

The initial setting of both mappings is suitable for normal a.out and core files. If either file is not of the kind expected then, for that file, b1 is set to 0, e1 is set to the maximum file size, and f1 is set to 0; in this way the whole file can be examined with no address translation.

In order for sdb to be used on large files, all appropriate values are kept as signed 32-bit integers.
Commands

The commands for examining data in the program are:

- **t** Print a stack trace of the terminated or halted program.
- **T** Print the top line of the stack trace.

**variable/clm**

Print the value of `variable` according to length `l` and format `m`. A numeric count `c` indicates that a region of memory, beginning at the address implied by `variable`, is to be displayed. The length specifiers are:

- `b` one byte
- `h` two bytes (half word)
- `l` four bytes (long word)

Legal values for `m` are:

- `c` character
- `d` decimal
- `u` decimal, unsigned
- `o` octal
- `x` hexadecimal
- `f` 32-bit single precision floating point
- `g` 64-bit double precision floating point
- `s` Assume `variable` is a string pointer and print characters starting at the address pointed to by the variable.
- `a` Print characters starting at the variable's address. This format may not be used with register variables.
- `p` pointer to procedure
- `i` disassemble machine-language instruction with addresses printed numerically and symbolically.
- `I` disassemble machine-language instruction with addresses just printed numerically.

Length specifiers are only effective with the `c`, `d`, `u`, `o` and `x` formats. Any of the specifiers, `c`, `l`, and `m`, may be omitted. If all are omitted, `sdb` chooses a length and a format suitable for the variable’s type as declared in the program. If `m` is specified, then this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier `c` tells `sdb` to display that many units of memory, beginning at the address of `variable`. The number of bytes in one such unit of memory is determined by the length specifier `l`, or if no length is given, by the size associated with the `variable`. If a count specifier is used for the `s` or `a` command, then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command `.f`.

The `sh(1)` metacharacters `*` and `?` may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, variables local to the current procedure and global variables are matched; if a procedure name is specified then only variables local to
that procedure are matched. To match only global variables, the form `:pattern` is used.

`linenumber?lm`  
`variable?lm`  
Print the value at the address from a.out or I space given by `linenumber` or `variable` (procedure name), according to the format `lm`. The default format is `'Y'.

`variable=lm`  
`linenumber=lm`  
`number=lm`  
Print the address of `variable` or `linenumber`, or the value of `number`, in the format specified by `lm`. If no format is given, then `lx` is used. The last variant of this command provides a convenient way to convert between decimal, octal and hexadecimal.

`variable=value`  
Set `variable` to the given `value`. The value may be a number, a character constant or a variable. The value must be well defined; expressions which produce more than one value, such as structures, are not allowed. Character constants are denoted `'character`. Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type double. Registers are viewed as integers. The `variable` may be an expression which indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type `int`. C conventions are used in any type conversions necessary to perform the indicated assignment.

`x`  
Print the machine registers and the current machine-language instruction.

`X`  
Print the current machine-language instruction.

The commands for examining source files are:

`e procedure`  
`e file-name`  
`e directory/`  
`e directory file-name`  
The first two forms set the current file to the file containing `procedure` or to `file-name`. The current line is set to the first line in the named procedure or file. Source files are assumed to be in `directory`. The default is the current working directory. The latter two forms change the value of `directory`. If no procedure, file name, or directory is given, the current procedure name and file name are reported.

`/regular expression/`  
Search forward from the current line for a line containing a string matching `regular expression` as in `ed(1)`. The trailing `/` may be deleted.

`?regular expression?`  
Search backward from the current line for a line containing a string matching `regular expression` as in `ed(1)`. The trailing `?` may be deleted.
SDB(1) (Extended Software Generation System Utilities) SDB(1)

p   Print the current line.

z   Print the current line followed by the next 9 lines. Set the current line to
the last line printed.

w   Window. Print the 10 lines around the current line.

donumber
Set the current line to the given line number. Print the new current line.

count+
   Advance the current line by count lines. Print the new current line.

count-
   Retreat the current line by count lines. Print the new current line.

The commands for controlling the execution of the source program are:

count r args

r   Run the program with the given arguments. The r command with no
arguments reuses the previous arguments to the program while the R com­
mand runs the program with no arguments. An argument beginning with
< or > causes redirection for the standard input or output, respectively. If
count is given, it specifies the number of breakpoints to be ignored.

linenumber c count

linenumber C count
Continue after a breakpoint or interrupt. If count is given, the program
will stop when count breakpoints have been encountered. The signal
which caused the program to stop is reactivated with the C command and
ignored with the c command. If a line number is specified then a tem­
porary breakpoint is placed at the line and execution is continued. The
breakpoint is deleted when the command finishes.

linenumber g count
Continue after a breakpoint with execution resumed at the given line. If
count is given, it specifies the number of breakpoints to be ignored.

s count
S count
Single step the program through count lines. If no count is given then the
program is run for one line. S is equivalent to s except it steps through
procedure calls.

i
I
Single step by one machine-language instruction. The signal which caused
the program to stop is reactivated with the I command and ignored with
the i command.

variable$m count
address$m count
Single step (as with s) until the specified location is modified with a new
value. If count is omitted, it is effectively infinity. Variable must be acces-
Possible from the current procedure. Since this command is done by software, it can be very slow.

```
level v
    Toggle verbose mode, for use when single stepping with S, s or m. If level is omitted, then just the current source file and/or subroutine name is printed when either changes. If level is 1 or greater, each C source line is printed before it is executed; if level is 2 or greater, each assembler statement is also printed. A v turns verbose mode off if it is on for any level.
```

```
k     Kill the program being debugged.
procedure(arg1,arg2,...)
procedure(arg1,arg2,...)/m
    Execute the named procedure with the given arguments. Arguments can be integer, character or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format m. If no format is given, it defaults to d. This facility is only available if the program was loaded with the -g option.
```

```
linenumber b commands
    Set a breakpoint at the given line. If a procedure name without a line number is given (e.g., "proc:"), a breakpoint is placed at the first line in the procedure even if it was not compiled with the -g option. If no linenumber is given, a breakpoint is placed at the current line. If no commands are given, execution stops just before the breakpoint and control is returned to sdb. Otherwise the commands are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons. If k is used as a command to execute at a breakpoint, control returns to sdb, instead of continuing execution.
```

```
B     Print a list of the currently active breakpoints.
linenumber d
    Delete a breakpoint at the given line. If no linenumber is given then the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from the standard input. If the line begins with a y or d then the breakpoint is deleted.
```

```
D     Delete all breakpoints.
L     Print the last executed line.
linenumber a
    Announce. If linenumber is of the form proc:number, the command effectively does a linenumber b l. If linenumber is of the form proc:, the command effectively does a proc: b T.
```

Miscellaneous commands:

```
!command
    The command is interpreted by sh(1).
```
new-line
If the previous command printed a source line, then advance the current
line by one line and print the new current line. If the previous command
displayed a memory location, then display the next memory location.

end-of-file character
Scroll. Print the next 10 lines of instructions, source or data depending on
which was printed last. The end-of-file character is usually control-D.

< filename
Read commands from filename until the end of file is reached, and then
continue to accept commands from standard input. When sdb is told to
display a variable by a command in such a file, the variable name is
displayed along with the value. This command may not be nested; < may
not appear as a command in a file.

M Print the address maps.

M ?/ [\*] b e f
Record new values for the address map. The arguments ? and / specify
the text and data maps, respectively. The first segment \((b1, e1, f1)\) is
changed unless \* is specified, in which case the second segment \((b2, e2, f2)\)
of the mapping is changed. If fewer than three values are given, the
remaining map parameters are left unchanged.

" string
Print the given string. The C escape sequences of the form \character are
recognized, where \character is a nonnumeric character.

q Exit the debugger.

The following commands also exist and are intended only for debugging the
debugger:

V Print the version number.
Q Print a list of procedures and files being debugged.
Y Toggle debug output.

FILES
a.out
core

SEE ALSO
cc(1), a.out(4), core(4), syms(4).
f77(1) in the FORTRAN Programming Language Manual.

WARNINGS
When sdb prints the value of an external variable for which there is no debug­
ing information, a warning is printed before the value. The size is assumed to
be int (integer).

Data which are stored in text sections are indistinguishable from functions.
Line number information in optimized functions is unreliable, and some infor­
mation may be missing.
BUGS

If a procedure is called when the program is *not* stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

`sdb` cannot print the value of an F77 parameter. It will erroneously print the address.

Tracebacks containing F77 subprograms with multiple entry points may print too many arguments in the wrong order, but their values are correct.

The range of an F77 array subscript is assumed to be 1 to $n$, where $n$ is the dimension corresponding to that subscript. This is only significant when the user omits a subscript, or uses `*` to indicate the full range. There is no problem in general with arrays having subscripts whose lower bounds are not 1.
NAME
size — print section sizes in bytes of common object files

SYNOPSIS

DESCRIPTION
The size command produces section size information in bytes for each loaded section in the common object files. The size of the text, data, and bss (uninitialized data) sections is printed, as well as the sum of the sizes of these sections. If an archive file is input to the size command the information for all archive members is displayed.

The -n option includes NOLOAD sections in the size.

The -f option produces full output, that is, it prints the size of every loaded section, followed by the section name in parentheses.

Numbers will be printed in decimal unless either the -o or the -x option is used, in which case they will be printed in octal or in hexadecimal, respectively.

The -V flag will supply the version information on the size command.

SEE ALSO
as(l), cc(l), ld(l), a.out(4), ar(4).

CAVEAT
Since the size of bss sections is not known until link-edit time, the size command will not give the true total size of pre-linked objects.

DIAGNOSTICS
size: name: cannot open
    if name cannot be read.

size: name: bad magic
    if name is not an appropriate common object file.
NAME

strip — strip symbol and line number information from a common object file

SYNOPSIS

```
strip [-l] [-x] [-b] [-r] [-V] filename ...
```

DESCRIPTION

The `strip` command strips the symbol table and line number information from common object files, including archives. Once this has been done, no symbolic debugging access will be available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using any of the following options:

- `-l` Strip line number information only; do not strip any symbol table information.
- `-x` Do not strip static or external symbol information.
- `-b` Same as the `-x` option, but also do not strip scoping information (e.g., beginning and end of block delimiters).
- `-r` Do not strip static or external symbol information, or relocation information.
- `-V` Print the version of the strip command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, `strip` will complain and terminate without stripping `filename` unless the `-r` option is used.

If the `strip` command is executed on a common archive file [see `ar(4)`] the archive symbol table will be removed. The archive symbol table must be restored by executing the `ar(1)` command with the `s` option before the archive can be link-edited by the `ld(1)` command. `strip` will produce appropriate warning messages when this situation arises.

`strip` is used to reduce the file storage overhead taken by the object file.

FILES

```
TMPDIR/strip* temporary files
```

`TMPDIR` is usually `/usr/tmp` but can be redefined by setting the environment variable `TMPDIR` [see `tempnam()` in `tmpnam(3S)`].

SEE ALSO

`ar(1), as(1), cc(1), ld(1), tmpnam(3S), a.out(4), ar(4)`.

DIAGNOSTICS

```
strip: name: cannot open
       if name cannot be read.

strip: name: bad magic
       if name is not an appropriate common object file.
```
strip: name: relocation entries present; cannot strip if name contains relocation entries and the -r flag is not used, the symbol table information cannot be stripped.
TIC(1M)  (Terminal Information Utilities)  TIC(1M)

NAME

tic – terminfo compiler

SYNOPSIS

tic [-v[n]] [-c] file

DESCRIPTION

tic translates a terminfo(4) file from the source format into the compiled format. The results are placed in the directory /usr/lib/terminfo. The compiled format is necessary for use with the library routines described in curses(3X).

- vn (verbose) output to standard error trace information showing tic’s progress. The optional integer n is a number from 1 to 10, inclusive, indicating the desired level of detail of information. If n is omitted, the default level is 1. If n is specified and greater than 1, the level of detail is increased.

- c only check file for errors. Errors in use= links are not detected.

file contains one or more terminfo(4) terminal descriptions in source format (see terminfo(4)). Each description in the file describes the capabilities of a particular terminal. When a use=entry-name field is discovered in a terminal entry currently being compiled, tic reads in the binary from /usr/lib/terminfo to complete the entry. (Entries created from file will be used first. If the environment variable TERMINFO is set, that directory is searched instead of /usr/lib/terminfo.) tic duplicates the capabilities in entry-name for the current entry, with the exception of those capabilities that explicitly are defined in the current entry.

If the environment variable TERMINFO is set, the compiled results are placed there instead of /usr/lib/terminfo.

FILES

/usr/lib/terminfo/?/*/ compiled terminal description data base

SEE ALSO


WARNINGS

Total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

Terminal names exceeding 14 characters will be truncated to 14 characters and a warning message will be printed.

When the -c option is used, duplicate terminal names will not be diagnosed; however, when -c is not used, they will be.

BUGS

To allow existing executables from the previous release of the UNIX System to continue to run with the compiled terminfo entries created by the new terminfo compiler, cancelled capabilities will not be marked as cancelled within the terminfo binary unless the entry name has a ‘+’ within it. (Such terminal names are only used for inclusion within other entries via a use= entry. Such names would not be used for real terminal names.)
For example:

\[4415+\text{nl}, \text{kf1@}, \text{kf2@}, \ldots\]
\[4415+\text{base}, \text{kf1=E0c}, \text{kf2=E0d}, \ldots\]
\[4415-\text{nl}\] 4415 terminal without keys,
\[\text{use}=4415+\text{nl}, \text{use}=4415+\text{base},\]

The above example works as expected; the definitions for the keys do not show up in the 4415-\text{nl} entry. However, if the entry 4415+\text{nl} did not have a plus sign within its name, the cancellations would not be marked within the compiled file and the definitions for the function keys would not be cancelled within 4415-\text{nl}.

**DIAGNOSTICS**

Most diagnostic messages produced by tic during the compilation of the source file are preceded with the approximate line number and the name of the terminal currently being worked on.

*mkdir ... returned bad status*
  The named directory could not be created.

*File does not start with terminal names in column one*
  The first thing seen in the file, after comments, must be the list of terminal names.

*Token after a \text{seek}(2) not \text{NAMES}*
  Somehow the file being compiled changed during the compilation.

*Not enough memory for \text{use} list element*
  or

*Out of memory*
  Not enough free memory was available (\text{malloc}(3) failed).

*Can't open ...*
  The named file could not be created.

*Error in writing ...*
  The named file could not be written to.

*Can't link ... to ...*
  A link failed.

*Error in re-reading compiled file ...*
  The compiled file could not be read back in.

*Premature EOF*
  The current entry ended prematurely.

*Backspaced off beginning of line*
  This error indicates something wrong happened within tic.

*Unknown Capability - "..."*
  The named invalid capability was found within the file.

*Wrong type used for capability "..."
  For example, a string capability was given a numeric value.*
Unknown token type
Tokens must be followed by '@' to cancel, ',' for booleans, '#' for numbers, or '=' for strings.

"...": bad term name
or
Line ...: Illegal terminal name - "...
Terminal names must start with a letter or digit
The given name was invalid. Names must not contain white space or slashes, and must begin with a letter or digit.

"...": terminal name too long.
An extremely long terminal name was found.

"...": terminal name too short.
A one-letter name was found.

"..." filename too long, truncating to "...
The given name was truncated to 14 characters due to UNIX file name length limitations.

"..." defined in more than one entry. Entry being used is "...
An entry was found more than once.

Terminal name "...", synonym for itself
A name was listed twice in the list of synonyms.

At least one synonym should begin with a letter.
At least one of the names of the terminal should begin with a letter.

Illegal character - "...
The given invalid character was found in the input file.

Newline in middle of terminal name
The trailing comma was probably left off of the list of names.

Missing comma
A comma was missing.

Missing numeric value
The number was missing after a numeric capability.

NULL string value
The proper way to say that a string capability does not exist is to cancel it.

Very long string found. Missing comma?
self-explanatory

Unknown option. Usage is:
An invalid option was entered.

Too many file names. Usage is:
self-explanatory

"..." non-existant or permission denied
The given directory could not be written into.
"..." is not a directory
self-explanatory

"...": Permission denied
access denied.

"...": Not a directory

tic wanted to use the given name as a directory, but it already exists as
a file

SYSTEM ERROR!! Fork failed!!!
A fork(2) failed.

Error in following up use-links. Either there is a loop in the links or they refer-
ence non-existant terminals. The following is a list of the entries involved:
A terminfo(4) entry with a use=name capability either referenced a non-
existant terminal called name or name somehow referred back to the
given entry.
NAME
tsort – topological sort

SYNOPSIS
tsort [file]

DESCRIPTION
The tsort command produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input file. If no file is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO
lorder(1).

DIAGNOSTICS
Odd data: there is an odd number of fields in the input file.
NAME
unget — undo a previous get of an SCCS file

SYNOPSIS
unget [-rSID] [-s] [-n] files

DESCRIPTION
unget undoes the effect of a get -e done prior to creating the intended new
delta. If a directory is named, unget behaves as though each file in the directory
were specified as a named file, except that non-SCCS files and unreadable files
are silently ignored. If a name of - is given, the standard input is read with
each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

- rSID Uniquely identifies which delta is no longer intended. (This
would have been specified by get as the "new delta"). The
use of this keyletter is necessary only if two or more out-
standing gets for editing on the same SCCS file were done by
the same person (login name). A diagnostic results if the
specified SID is ambiguous, or if it is necessary and omitted
on the command line.

-s Suppresses the printout, on the standard output, of the
intended delta's SID.

-n Causes the retention of the gotten file which would normally
be removed from the current directory.

SEE ALSO
delta(1), get(1), sact(1).

DIAGNOSTICS
Use help(1) for explanations.
NAME
val - validate SCCS file

SYNOPSIS
val -
val [-s] [-rSID] [-mname] [-ytype] files

DESCRIPTION
val determines if the specified file is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to val may appear in any order. The arguments consist of keyletter arguments, which begin with a -, and named files.

val has a special argument, -, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

-s The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

-rSID The argument value SID (SCCS IDentification String) is an SCCS delta number. A check is made to determine if the SID is ambiguous (e. g., r1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (e. g., r1.0 or r1.1.0 are invalid because neither case can exist as a valid delta number). If the SID is valid and not ambiguous, a check is made to determine if it actually exists.

-mname The argument value name is compared with the %M% keyword in file.

-ytype The argument value type is compared with the SCCS %Y% keyword in file.

The 8-bit code returned by val is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = SID is invalid or ambiguous;
bit 5 = SID does not exist;
bit 6 = %Y%, -y mismatch;
bit 7 = %M%, -m mismatch;
Note that `val` can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned — a logical OR of the codes generated for each command line and file processed.

**SEE ALSO**

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

help(1) in the *User's Reference Manual*.

**DIAGNOSTICS**

Use `help(1)` for explanations.

**BUGS**

`val` can process up to 50 files on a single command line. Any number above 50 will produce a core dump.
NAME
vc - version control

SYNOPSIS
vc [-a] [-t] [-c char] [-s] [keyword=value ... keyword=value]

DESCRIPTION
The vc command copies lines from the standard input to the standard output under control of its arguments and control statements encountered in the standard input. In the process of performing the copy operation, user declared keywords may be replaced by their string value when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as vc command arguments.

A control statement is a single line beginning with a control character, except as modified by the -t keyletter (see below). The default control character is colon (:), except as modified by the -c keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with ed(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The -a keyletter (see below) forces replacement of keywords in all text lines and not just in vc control statements. The -a keyletter (see below) forces replacement of keywords in all lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter Arguments
-a Forces replacement of keywords surrounded by control characters with their assigned value in all text lines and not just in vc statements.
-t All characters from the beginning of a line up to and including the first tab character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the tab are discarded.
-c char Specifies a control character to be used in place of :.
-s Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements
:dcl keyword[, ... , keyword]
Used to declare keywords. All keywords must be declared.
:asg keyword=value
Used to assign values to keywords. An asg statement overrides the assignment for the corresponding keyword on the vc command line and all previous asg's for that keyword. Keywords declared, but not assigned values have null values.

:if condition
::
:end
Used to skip lines of the standard input. If the condition is true all lines between the if statement and the matching end statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening if statements and matching end statements are recognized solely for the purpose of maintaining the proper if-end matching.
The syntax of a condition is:

<cond> ::= [ "not" ] <or>
<or> ::= <and> | <and> "|" <or>
<and> ::= <exp> | <exp> "&" <and>
<exp> ::= "(" <or> ")" | <value> <op> <value>
<op> ::= "-" | "!=" | "<" | ">
[value] ::= <arbitrary ASCII string> | <numeric string>

The available operators and their meanings are:

=        equal
!=       not equal
&        and
|         or
>         greater than
<         less than
()       used for logical groupings
not      may only occur immediately after the if, and
         when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

= != > <      all of equal precedence
&             
|             

Parentheses may be used to alter the order of precedence.
Values must be separated from operators or parentheses by at least one blank or tab.
::text
  Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the -a keyletter.

:on
:off
  Turn on or off keyword replacement on all lines.

:ctl char
  Change the control character to char.

:msg message
  Prints the given message on the diagnostic output.

:err message
  Prints the given message followed by:
  ERROR: err statement on line ... (915)
  on the diagnostic output. vc halts execution, and returns an exit code of 1.

SEE ALSO

EXIT CODES
  0 — normal
  1 — any error
  Use help(1) for explanations.
NAME
what — identify SCCS files

SYNOPSIS
what [-s] files

DESCRIPTION
what searches the given files for all occurrences of the pattern that get(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first -, >, new-line, \, or null character. For example, if the C program in file f.c contains

    char ident[] = "@(#)identification information";

and f.c is compiled to yield f.o and a.out, then the command

    what f.c f.o a.out

will print

    f.c:
    identification information

    f.o:
    identification information

    a.out:
    identification information

what is intended to be used in conjunction with the command get(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

    -s    Quit after finding the first occurrence of pattern in each file.

SEE ALSO
get(1),

DIAGNOSTICS
Exit status is 0 if any matches are found, otherwise 1. Use help(1) for explanations.

BUGS
It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.
NAME
wtinit — object downloader for the 5620 DMD terminal

SYNOPSIS
/usr/lib/layersys/wtinit [-d] [-p] file

DESCRIPTION
The wtinit utility downloads the named file for execution in the AT&T Teletype 5620 DMD terminal connected to its standard output. file must be a DMD object file. wtinit performs all necessary bootstrap and protocol procedures.

There are two options.

-d     Prints out the sizes of the text, data, and bss portions of the downloaded file on standard error.

-p     Prints the download protocol statistics and a trace on standard error.

The environment variable JPATH is the analog of the shell’s PATH variable to define a set of directories in which to search for file.

If the environment variable DMDLOAD has the value hex, wtinit will use a hexadecimal download protocol that uses only printable characters.

Terminal Feature Packages for specific versions of AT&T windowing terminals will include terminal-specific versions of wtinit under those installation subdirectories. /usr/lib/layersys/wtinit is used for layers(1) initialization only when no Terminal Feature Package is in use.

DIAGNOSTICS
Returns 0 upon successful completion, 1 otherwise.

WARNING
Standard error should be redirected when using the -d or -p options.

SEE ALSO
NAME
xtd – extract and print xt driver link structure

SYNOPSIS
xtd [-f] [-n ...]

DESCRIPTION
The `xtd` command is a debugging tool for the `xt(7)` driver. It performs an 
`XTIOCDATA ioctl(2)` call on its standard input file to extract the Link data struc-
ture for the attached group of channels. This call will fail if data extraction has 
not been configured in the driver or the standard input is not attached to an 
`xt(7)` channel. The data are printed one item per line on the standard output. 
The output should probably be formatted via `pr -3`.

The optional flags affect output as follows:

- `-n` $n$ is a number in the range 0 to 7. Channel $n$ is included in the list of 
  channels to be printed. The default prints all channels, whereas the 
  occurrence of one or more channel numbers implies a subset.

- `-f` Causes a “formfeed” character to be put out at the end of the output, 
  for the benefit of page-display programs.

DIAGNOSTICS
Returns 0 upon successful completion, 1 otherwise.

SEE ALSO
NAME
xts — extract and print xt driver statistics

SYNOPSIS
xts [-f]

DESCRIPTION
The xts command is a debugging tool for the xt(7) driver. It performs an
XTIOCSTATS ioctl(2) call on its standard input file to extract the accumulated
statistics for the attached group of channels. This call will fail if statistics have
not been configured in the driver or the standard input is not attached to an
xt(7) channel. The statistics are printed one item per line on the standard
output.

-f Causes a "formfeed" character to be put out at the end of the output,
for the benefit of page-display programs.

DIAGNOSTICS
Returns 0 upon successful completion, 1 otherwise.

SEE ALSO
NAME
xtt – extract and print xt driver packet traces

SYNOPSIS
xtt [-f] [-o]

DESCRIPTION
The xtt command is a debugging tool for the xt(7) driver. It performs an XTIOC­
TRACE ioctl(2) call on its standard input file to turn on tracing and extract the
circular packet trace buffer for the attached group of channels. This call will fail
if tracing has not been configured in the driver, or the standard input is not
attached to an xt(7) channel. The packets are printed on the standard output.

The optional flags are:
-f Causes a "formfeed" character to be put out at the end of the output, for
the benefit of page-display programs.
-o Turns off further driver tracing.

DIAGNOSTICS
Returns 0 upon successful completion, 1 otherwise.

NOTE
If driver tracing has not been turned on for the terminal session by invoking
layers(1) with the -t option, xtt will not generate any output the first time it is
executed.

SEE ALSO
NAME
yacc — yet another compiler-compiler

SYNOPSIS
yacc [ -vdlt ] grammar

DESCRIPTION
The `yacc` command converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, `y.tab.c`, must be compiled by the C compiler to produce a program `yyparse`. This program must be loaded with the lexical analyzer program, `yylex`, as well as `main` and `yyerror`, an error handling routine. These routines must be supplied by the user; `lex(1)` is useful for creating lexical analyzers usable by `yacc`.

If the `−v` flag is given, the file `y.output` is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the `−d` flag is used, the file `y.tab.h` is generated with the `#define` statements that associate the `yacc`-assigned “token codes” with the user-declared “token names”. This allows source files other than `y.tab.c` to access the token codes.

If the `−l` flag is given, the code produced in `y.tab.c` will not contain any `#line` constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in `y.tab.c` under conditional compilation control. By default, this code is not included when `y.tab.c` is compiled. However, when `yacc`‘s `−t` option is used, this debugging code will be compiled by default. Independent of whether the `−t` option was used, the runtime debugging code is under the control of `YYDEBUG`, a preprocessor symbol. If `YYDEBUG` has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

FILES
y.output
y.tab.c
y.tab.h defines for token names
yacc.tmp,
yacc.debug, yacc.acts temporary files
/usr/lib/yaccpar parser prototype for C programs

SEE ALSO
lex(1).
`Programmer's Guide.`
DIAGNOSTICS
The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the y.output file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

CAVEAT
Because file names are fixed, at most one yacc process can be active in a given directory at a given time.
NAME
intro — introduction to system calls and error numbers

SYNOPSIS
#include <errno.h>

DESCRIPTION
This section describes all of the system calls. Most of these calls have one or
more error returns. An error condition is indicated by an otherwise impossible
returned value. This is almost always -1 or the NULL pointer; the individual
descriptions specify the details. An error number is also made available in the
external variable errno. Errno is not cleared on successful calls, so it should be
tested only after an error has been indicated.

Each system call description attempts to list all possible error numbers. The fol­
lowing is a complete list of the error numbers and their names as defined in
<errno.h>.

1 EPERM Not owner
Typically this error indicates an attempt to modify a file in some way
forbidden except to its owner or super-user. It is also returned for
attempts by ordinary users to do things allowed only to the super-user.

2 ENOENT No such file or directory
This error occurs when a file name is specified and the file should exist
but doesn’t, or when one of the directories in a path name does not
exist.

3 ESRCH No such process
No process can be found corresponding to that specified by pid in kill(2)
or ptrace(2).

4 EINTR Interrupted system call
An asynchronous signal (such as interrupt or quit), which the user has
elected to catch, occurred during a system call. If execution is resumed
after processing the signal, it will appear as if the interrupted system call
returned this error condition.

5 EIO I/O error
Some physical I/O error has occurred. This error may in some cases
occur on a call following the one to which it actually applies.

6 ENXIO No such device or address
I/O on a special file refers to a subdevice which does not exist, or
beyond the limits of the device. It may also occur when, for example, a
tape drive is not on-line or no disk pack is loaded on a drive.

7 E2BIG Arg list too long
An argument list longer than 5,120 bytes is presented to a member of
the exec(2) family.

8 ENOEXEC Exec format error
A request is made to execute a file which, although it has the
appropriate permissions, does not start with a valid magic number [see
a.out(4)].
9 EBADF Bad file number
Either a file descriptor refers to no open file, or a read(2) [respectively, write(2)] request is made to a file which is open only for writing (respectively, reading).

10 ECHILD No child processes
A wait was executed by a process that had no existing or unwaited-for child processes.

11 EAGAIN No more processes
A fork failed because the system's process table is full or the user is not allowed to create any more processes. Or a system call failed because of insufficient memory or swap space.

12 ENOMEM Not enough space
During an exec(2), brk(2), or sbrk(2), a program asks for more space than the system is able to supply. This may not be a temporary condition; the maximum space size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers, or if there is not enough swap space during a fork(2). If this error occurs on a resource associated with Remote File Sharing (RFS), it indicates a memory depletion which may be temporary, dependent on system activity at the time the call was invoked.

13 EACCESS Permission denied
An attempt was made to access a file in a way forbidden by the protection system.

14 EFAULT Bad address
The system encountered a hardware fault in attempting to use an argument of a system call.

15 ENOTBLK Block device required
A non-block file was mentioned where a block device was required, e.g., in mount(2).

16 EBUSY Device or resource busy
An attempt was made to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment). It will also occur if an attempt is made to enable accounting when it is already enabled. The device or resource is currently unavailable.

17 EEXIST File exists
An existing file was mentioned in an inappropriate context, e.g., link(2).

18 EXDEV Cross-device link
A link to a file on another device was attempted.

19 ENODEV No such device
An attempt was made to apply an inappropriate system call to a device; e.g., read a write-only device.
20 ENOTDIR  Not a directory
   A non-directory was specified where a directory is required, for example
   in a path prefix or as an argument to chdir(2).

21 EISDIR  Is a directory
   An attempt was made to write on a directory.

22 EINVAL  Invalid argument
   Some invalid argument (e.g., dismounting a non-mounted device; men­
   tioning an undefined signal in signal(2) or kill(2); reading or writing a file
   for which lseek(2) has generated a negative pointer).  Also set by the
   math functions described in the (3M) entries of this manual.

23 ENFILE  File table overflow
   The system file table is full, and temporarily no more opens can be
   accepted.

24 EMFILE  Too many open files
   No process may have more than NOFILES (default 20) descriptors open
   at a time.

25 ENOTTY  Not a character device (or) Not a typewriter
   An attempt was made to ioctl(2) a file that is not a special character
   device.

26 ETXTBSY  Text file busy
   An attempt was made to execute a pure-procedure program that is
   currently open for writing.  Also an attempt to open for writing or to
   remove a pure-procedure program that is being executed.

27 EFBIG  File too large
   The size of a file exceeded the maximum file size or ULIMIT [see
   ulimit(2)].

28 ENOSPC  No space left on device
   During a write(2) to an ordinary file, there is no free space left on the
   device.  In fcntl(2), the setting or removing of record locks on a file
   cannot be accomplished because there are no more record entries left on
   the system.

29 ESPIPE  Illegal seek
   An lseek(2) was issued to a pipe.

30 EROFS  Read-only file system
   An attempt to modify a file or directory was made on a device mounted
   read-only.

31 EMLINK  Too many links
   An attempt to make more than the maximum number of links (1000) to
   a file.

32 EPIPE  Broken pipe
   A write on a pipe for which there is no process to read the data.  This
   condition normally generates a signal; the error is returned if the signal
   is ignored.
33 EDOM Math argument
   The argument of a function in the math package (3M) is out of the
domain of the function.

34 ERANGE Result too large
   The value of a function in the math package (3M) is not representable
within machine precision.

35 ENOMSG No message of desired type
   An attempt was made to receive a message of a type that does not exist
on the specified message queue [see msgop(2)].

36 EIDRM Identifier removed
   This error is returned to processes that resume execution due to the
removal of an identifier from the file system’s name space [see
msgctl(2), semctl(2), and shmctl(2)].

37-44 Reserved numbers

45 EDEADLK Deadlock
   A deadlock situation was detected and avoided. This error pertains to
file and record locking.

46 ENOLCK No lock
   In fcntl(2) the setting or removing of record locks on a file cannot be
accomplished because there are no more record entries left on the
system.

60 ENOSTR Not a stream
   A putmsg(2) or getmsg(2) system call was attempted on a file descriptor
that is not a STREAMS device.

62 ETIME Stream ioctl timeout
   The timer set for a STREAMS ioctl(2) call has expired. The cause of this
error is device specific and could indicate either a hardware or software
failure, or perhaps a timeout value that is too short for the specific
operation. The status of the ioctl(2) operation is indeterminate.

63 ENOSR No stream resources
   During a STREAMS open(2), either no STREAMS queues or no STREAMS
head data structures were available.

64 ENONET Machine is not on the network
   This error is Remote File Sharing (RFS) specific. It occurs when users try
to advertise, unadvertise, mount, or unmount remote resources while the
machine has not done the proper startup to connect to the network.

65 ENOPKG No package
   This error occurs when users attempt to use a system call from a
package which has not been installed.

66 EREMOTE Resource is remote
   This error is RFS specific. It occurs when users try to advertise a resource
which is not on the local machine, or try to mount/unmount a device
(or pathname) that is on a remote machine.
67 ENOLINK Virtual circuit is gone
   This error is RFS specific. It occurs when the link (virtual circuit) connecting to a remote machine is gone.

68 EADV Advertise error
   This error is RFS specific. It occurs when users try to advertise a resource which has been advertised already, or try to stop the RFS while there are resources still advertised, or try to force unmount a resource when it is still advertised.

69 ESRMNT Srmount error
   This error is RFS specific. It occurs when users try to stop RFS while there are resources still mounted by remote machines.

70 ECOMM Communication error
   This error is RFS specific. It occurs when trying to send messages to remote machines but no virtual circuit can be found.

71 EPROTO Protocol error
   Some protocol error occurred. This error is device specific, but is generally not related to a hardware failure.

74 EMULTIHO P Multihop attempted
   This error is RFS specific. It occurs when users try to access remote resources which are not directly accessible.

77 EBADMSG Bad message
   During a read(2), getmsg(2), or ioctl(2) I_RECVFD system call to a STREAMS device, something has come to the head of the queue that can't be processed. That something depends on the system call:
   read(2) - control information or a passed file descriptor.
   getmsg(2) - passed file descriptor.
   ioctl(2) - control or data information.

83 ELIBACC Cannot access a needed shared library
   Trying to exec(2) an a.out that requires a shared library (to be linked in) and the shared library doesn't exist or the user doesn't have permission to use it.

84 ELIBBAD Accessing a corrupted shared library
   Trying to exec(2) an a.out that requires a shared library (to be linked in) and exec(2) could not load the shared library. The shared library is probably corrupted.

85 ELIBSCN .lib section in a.out corrupted
   Trying to exec(2) an a.out that requires a shared library (to be linked in) and there was erroneous data in the .lib section of the a.out. The .lib section tells exec(2) what shared libraries are needed. The a.out is probably corrupted.

86 ELIBMAX Attempting to link in more shared libraries than system limit
   Trying to exec(2) an a.out that requires more shared libraries (to be linked in) than is allowed on the current configuration of the system. See the System Administrator's Guide.
87 ELIBEXEC Cannot exec a shared library directly
Trying to exec(2) a shared library directly. This is not allowed.

DEFINITIONS

Process ID Each active process in the system is uniquely identified by a positive integer called a process ID. The range of this ID is from 1 to 30,000.

Parent Process ID A new process is created by a currently active process [see fork(2)]. The parent process ID of a process is the process ID of its creator.

Process Group ID Each active process is a member of a process group that is identified by a positive integer called the process group ID. This ID is the process ID of the group leader. This grouping permits the signaling of related processes [see kill(2)].

Tty Group ID Each active process can be a member of a terminal group that is identified by a positive integer called the tty group ID. This grouping is used to terminate a group of related processes upon termination of one of the processes in the group [see exit(2) and signal(2)].

Real User ID and Real Group ID Each user allowed on the system is identified by a positive integer (0 to 65535) called a real user ID.
Each user is also a member of a group. The group is identified by a positive integer called the real group ID.
An active process has a real user ID and real group ID that are set to the real user ID and real group ID, respectively, of the user responsible for the creation of the process.

Effective User ID and Effective Group ID An active process has an effective user ID and an effective group ID that are used to determine file access permissions (see below). The effective user ID and effective group ID are equal to the process's real user ID and real group ID respectively, unless the process or one of its ancestors evolved from a file that had the set-user-ID bit or set-group ID bit set [see exec(2)].

Super-user A process is recognized as a super-user process and is granted special privileges, such as immunity from file permissions, if its effective user ID is 0.

Special Processes The processes with a process ID of 0 and a process ID of 1 are special processes and are referred to as proc0 and proc1.
**Proc0** is the scheduler. **Proc1** is the initialization process (init). Proc1 is the ancestor of every other process in the system and is used to control the process structure.

**File Descriptor** A file descriptor is a small integer used to do I/O on a file. The value of a file descriptor is from 0 to (NOFILES - 1). A process may have no more than NOFILES file descriptors open simultaneously. A file descriptor is returned by system calls such as `open(2)`, or `pipe(2)`. The file descriptor is used as an argument by calls such as `read(2)`, `write(2)`, `ioctl(2)`, and `close(2)`.

**File Name** Names consisting of 1 to 14 characters may be used to name an ordinary file, special file or directory.

These characters may be selected from the set of all character values excluding \0 (null) and the ASCII code for / (slash).

Note that it is generally unwise to use *, ?, [, or ] as part of file names because of the special meaning attached to these characters by the shell [see `sh(1)`]. Although permitted, the use of unprintable characters in file names should be avoided.

**Path Name and Path Prefix** A path name is a null-terminated character string starting with an optional slash (/), followed by zero or more directory names separated by slashes, optionally followed by a file name.

If a path name begins with a slash, the path search begins at the root directory. Otherwise, the search begins from the current working directory.

A slash by itself names the root directory.

Unless specifically stated otherwise, the null path name is treated as if it named a non-existent file.

**Directory** Directory entries are called links. By convention, a directory contains at least two links, . and .., referred to as dot and dot-dot respectively. Dot refers to the directory itself and dot-dot refers to its parent directory.

**Root Directory and Current Working Directory** Each process has associated with it a concept of a root directory and a current working directory for the purpose of resolving path name searches. The root directory of a process need not be the root directory of the root file system.

**File Access Permissions** Read, write, and execute/search permissions on a file are granted to a process if one or more of the following are true:

- The effective user ID of the process is super-user.
- The effective user ID of the process matches the user ID of the owner of the file and the appropriate access bit of the "owner" portion (0700) of the file mode is set.
The effective user ID of the process does not match the user ID of the owner of the file, and the effective group ID of the process matches the group of the file and the appropriate access bit of the "group" portion (0070) of the file mode is set.

The effective user ID of the process does not match the user ID of the owner of the file, and the effective group ID of the process does not match the group ID of the file, and the appropriate access bit of the "other" portion (0007) of the file mode is set.

Otherwise, the corresponding permissions are denied.

Message Queue Identifier A message queue identifier (msqid) is a unique positive integer created by a msgget(2) system call. Each msqid has a message queue and a data structure associated with it. The data structure is referred to as msqid_ds and contains the following members:

```c
struct ipc_perm msg_perm;
struct msg *msg_first;
struct msg *msg_last;
ushort msg_cbytes;
ushort msg_qnum;
ushort msg_qbytes;
ushort msg_seqid;
ushort msg_lpid;
time_t msg_stime;
time_t msg_rtime;
time_t msg_ctime;
```

msg_perm is an ipc_perm structure that specifies the message operation permission (see below). This structure includes the following members:

```c
ushort cuid; /* creator user id */
ushort cgid; /* creator group id */
ushort uid; /* user id */
ushort gid; /* group id */
ushort mode; /* r/w permission */
ushort seq; /* slot usage sequence # */
key_t key; /* key */
```

msg *msg_first
is a pointer to the first message on the queue.

msg *msg_last
is a pointer to the last message on the queue.

msg_cbytes
is the current number of bytes on the queue.

msg_qnum
is the number of messages currently on the queue.

msg_qbytes
is the maximum number of bytes allowed on the queue.
**msg_lspid**
is the process id of the last process that performed a `msgsnd` operation.

**msg_lrp0id**
is the process id of the last process that performed a `msgrcv` operation.

**msg_stime**
is the time of the last `msgsnd` operation.

**msg_rtime**
is the time of the last `msgrcv` operation.

**msg_ctime**
is the time of the last `msgctl(2)` operation that changed a member of the above structure.

**Message Operation Permissions** In the `msgop(2)` and `msgctl(2)` system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed, interpreted as follows:

- 00400  Read by user
- 00200  Write by user
- 00040  Read by group
- 00020  Write by group
- 00004  Read by others
- 00002  Write by others

Read and write permissions on a msqid are granted to a process if one or more of the following are true:

- The effective user ID of the process is super-user.
- The effective user ID of the process matches `msg_perm.cuid` or `msg_perm.uid` in the data structure associated with `msqid` and the appropriate bit of the "user" portion (0600) of `msg_perm.mode` is set.
- The effective group ID of the process matches `msg_perm.cgid` or `msg_perm.gid` and the appropriate bit of the "group" portion (060) of `msg_perm.mode` is set.
- The appropriate bit of the "other" portion (006) of `msg_perm.mode` is set.

Otherwise, the corresponding permissions are denied.

**Semaphore Identifier** A semaphore identifier (semid) is a unique positive integer created by a `semget(2)` system call. Each semid has a set of semaphores and a data structure associated with it. The data structure is referred to as `semid_ds` and contains the following members:

```c
struct ipc_perm sem_perm; /* operation permission struct */
struct sem *sem_base; /* ptr to first semaphore in set */
ushort sem_nsems; /* number of sems in set */
time_t sem_otime; /* last operation time */
time_t sem_ctime; /* last change time */
/* Times measured in secs since */
/* 00:00:00 GMT, Jan. 1, 1970 */
```
sem_perm is an ipc_perm structure that specifies the semaphore operation permission (see below). This structure includes the following members:

```c
ushort uid;  /* user id */
ushort gid;  /* group id */
ushort cuid; /* creator user id */
ushort cgid; /* creator group id */
ushort mode; /* r/w/a permission */
ushort seq;  /* slot usage sequence number */
key_t key;  /* key */
```

sem_nsems is equal to the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as a sem_num. Sem_num values run sequentially from 0 to the value of sem_nsems minus 1.

semmtime is the time of the last semop(2) operation.

semctime is the time of the last semctl(2) operation that changed a member of the above structure.

A semaphore is a data structure called sem that contains the following members:

```c
ushort semval;  /* semaphore value */
short sempid;  /* pid of last operation */
ushort semncnt; /* # awaiting semval > cval */
ushort semzcnt; /* # awaiting semval = 0 */
```

semval is a non-negative integer which is the actual value of the semaphore.

sempid is equal to the process ID of the last process that performed a semaphore operation on this semaphore.

semncnt is a count of the number of processes that are currently suspended awaiting this semaphore’s semval to become greater than its current value.

semzcnt is a count of the number of processes that are currently suspended awaiting this semaphore’s semval to become zero.

**Semaphore Operation Permissions** In the semop(2) and semctl(2) system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed interpreted as follows:

<table>
<thead>
<tr>
<th>Token</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>00400</td>
<td>Read by user</td>
</tr>
<tr>
<td>00200</td>
<td>Alter by user</td>
</tr>
<tr>
<td>00040</td>
<td>Read by group</td>
</tr>
</tbody>
</table>
Read and alter permissions on a semid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches sem_perm.cuid or sem_perm.uid in the data structure associated with semid and the appropriate bit of the “user” portion (0600) of sem_perm.mode is set.

The effective group ID of the process matches sem_perm.cgid or sem_perm.gid and the appropriate bit of the “group” portion (060) of sem_perm.mode is set.

The appropriate bit of the “other” portion (006) of sem_perm.mode is set.

Otherwise, the corresponding permissions are denied.

Shared Memory Identifier A shared memory identifier (shmid) is a unique positive integer created by a shmget(2) system call. Each shmid has a segment of memory (referred to as a shared memory segment) and a data structure associated with it. (Note that these shared memory segments must be explicitly removed by the user after the last reference to them is removed.) The data structure is referred to as shmid_ds and contains the following members:

```
struct ipc_perm shm_perm; /* operation permission struct */
int shm_segsz; /* size of segment */
struct region *shm_reg; /* ptr to region structure */
char pad[4]; /* for swap compatibility */
ushort shm_lpid; /* pid of last operation */
ushort shm_cpid; /* creator pid */
ushort shm_natch; /* number of current attaches */
ushort shm_cnattch; /* used only for shminfo */
time_t shm_atime; /* last attach time */
time_t shm_dtime; /* last detach time */
time_t shm_ctime; /* last change time */
/* Times measured in secs since */
/* 00:00:00 GMT, Jan. 1, 1970 */
```

shm_perm is an ipc_perm structure that specifies the shared memory operation permission (see below). This structure includes the following members:

```
ushort cuid; /* creator user id */
ushort cgid; /* creator group id */
ushort uid; /* user id */
ushort gid; /* group id */
ushort mode; /* r/w permission */
ushort seq; /* slot usage sequence # */
key_t key; /* key */
```
shm_segsz
specifies the size of the shared memory segment in bytes.

shm_cpid
is the process id of the process that created the shared memory identifier.

shm_lpid
is the process id of the last process that performed a shmop(2) operation.

shm_nattch
is the number of processes that currently have this segment attached.

shm_atime
is the time of the last shmat(2) operation,

shm_dtime
is the time of the last shmdt(2) operation.

shm_ctime
is the time of the last shmctl(2) operation that changed one of the members of the above structure.

**Shared Memory Operation Permissions**
In the shmop(2) and shmctl(2) system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed interpreted as follows:

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00400</td>
<td>Read by user</td>
</tr>
<tr>
<td>00200</td>
<td>Write by user</td>
</tr>
<tr>
<td>00040</td>
<td>Read by group</td>
</tr>
<tr>
<td>00020</td>
<td>Write by group</td>
</tr>
<tr>
<td>00004</td>
<td>Read by others</td>
</tr>
<tr>
<td>00002</td>
<td>Write by others</td>
</tr>
</tbody>
</table>

Read and write permissions on a shmid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **shm_perm.cuid** or **shm_perm.uid** in the data structure associated with **shmid** and the appropriate bit of the "user" portion (0600) of **shm_perm.mode** is set.

The effective group ID of the process matches **shm_perm.ggid** or **shm_perm.gid** and the appropriate bit of the "group" portion (06) of **shm_perm.mode** is set.

The appropriate bit of the "other" portion (06) of **shm_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

**STREAMS** A set of kernel mechanisms that support the development of network services and data communication **drivers**. It defines interface standards for char-
acter input/output within the kernel and between the kernel and user level processes. The STREAMS mechanism is composed of utility routines, kernel facilities and a set of data structures.

**Stream** A stream is a full-duplex data path within the kernel between a user process and driver routines. The primary components are a *stream head*, a *driver* and zero or more *modules* between the *stream head* and *driver*. A *stream* is analogous to a Shell pipeline except that data flow and processing are bidirectional.

**Stream Head** In a *stream*, the *stream head* is the end of the *stream* that provides the interface between the *stream* and a user process. The principle functions of the *stream head* are processing STREAMS-related system calls, and passing data and information between a user process and the *stream*.

**Driver** In a *stream*, the *driver* provides the interface between peripheral hardware and the *stream*. A *driver* can also be a pseudo-driver, such as a multiplexor or log *driver* [see log(7)], which is not associated with a hardware device.

**Module** A module is an entity containing processing routines for input and output data. It always exists in the middle of a *stream*, between the stream's head and a *driver*. A *module* is the STREAMS counterpart to the commands in a Shell pipeline except that a module contains a pair of functions which allow independent bidirectional (downstream and upstream) data flow and processing.

**Downstream** In a *stream*, the direction from *stream head* to *driver*.

**Upstream** In a *stream*, the direction from *driver* to *stream head*.

**Message** In a *stream*, one or more blocks of data or information, with associated STREAMS control structures. *Messages* can be of several defined types, which identify the *message* contents. *Messages* are the only means of transferring data and communicating within a *stream*.

**Message Queue** In a *stream*, a linked list of *messages* awaiting processing by a *module* or *driver*.

**Read Queue** In a *stream*, the *message queue* in a *module* or *driver* containing *messages* moving upstream.

**Write Queue** In a *stream*, the *message queue* in a *module* or *driver* containing *messages* moving downstream.
**Multiplexor** A multiplexor is a driver that allows *streams* associated with several user processes to be connected to a single *driver*, or several *drivers* to be connected to a single user process. STREAMS does not provide a general multiplexing *driver*, but does provide the facilities for constructing them, and for connecting multiplexed configurations of *streams*.

**SEE ALSO**
intro(3).
NAME
access — determine accessibility of a file

SYNOPSIS

    int access (path, amode)
    char *path;
    int amode;

DESCRIPTION
Path points to a path name naming a file. access checks the named file for
accessibility according to the bit pattern contained in amode, using the real user
ID in place of the effective user ID and the real group ID in place of the effective
group ID. The bit pattern contained in amode is constructed as follows:

04    read
02    write
01    execute (search)
00    check existence of file

Access to the file is denied if one or more of the following are true:

[ENOTDIR] A component of the path prefix is not a directory.
[ENOENT] Read, write, or execute (search) permission is
requested for a null path name.

[ENOENT] The named file does not exist.

[EACCES] Search permission is denied on a component of the
path prefix.

[EROFS] Write access is requested for a file on a read-only
file system.

[ETXTBSY] Write access is requested for a pure procedure
(shared text) file that is being executed.

[EACCES] Permission bits of the file mode do not permit
the requested access.

[EFAULT] Path points outside the allocated address
space for the process.

[EINVAL] A signal was caught during the access
system call.

[ENOLINK] Path points to a remote machine and the link
to that machine is no longer active.

[EMULTIHOP] Components of path require hopping to multiple
remote machines.

The owner of a file has permission checked with respect to the "owner" read,
write, and execute mode bits. Members of the file's group other than the owner
have permissions checked with respect to the "group" mode bits, and all others
have permissions checked with respect to the "other" mode bits.

SEE ALSO

chmod(2), stat(2).

DIAGNOSTICS
If the requested access is permitted, a value of 0 is returned. Otherwise, a value
of −1 is returned and errno is set to indicate the error.
NAME
acct — enable or disable process accounting

SYNOPSIS
int acct (path)
char *path;

DESCRIPTION
acct is used to enable or disable the system process accounting routine. If the
routine is enabled, an accounting record will be written on an accounting file for
each process that terminates. Termination can be caused by one of two things:
an exit call or a signal [see exit(2) and signal(2)]. The effective user ID of the cal-
ing process must be super-user to use this call.

path points to a pathname naming the accounting file. The accounting file
format is given in acct(4).

The accounting routine is enabled if path is non-zero and no errors occur during
the system call. It is disabled if path is zero and no errors occur during the
system call.

acct will fail if one or more of the following are true:
[EPERM] The effective user of the calling process is not super-user.
[EBUSY] An attempt is being made to enable accounting when it is
already enabled.
[ENOTDIR] A component of the path prefix is not a directory.
[ENOENT] One or more components of the accounting file path name do
not exist.
[EACCES] The file named by path is not an ordinary file.
[EROFS] The named file resides on a read-only file system.
[EFAULT] Path points to an illegal address.

SEE ALSO
exit(2), signal(2), acct(4).

DIAGNOSTICS
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is
returned and errno is set to indicate the error.
NAME
alarm — set a process alarm clock

SYNOPSIS
unsigned alarm (sec)
unsigned sec;

DESCRIPTION
alarm instructs the alarm clock of the calling process to send the signal SIGALRM
to the calling process after the number of real time seconds specified by sec have
elapsed [see signal(2)].

Alarm requests are not stacked; successive calls reset the alarm clock of the cal­
ing process.
If sec is 0, any previously made alarm request is canceled.

SEE ALSO
pause(2), signal(2), sigpause(2), sigset(2).

DIAGNOSTICS
alarm returns the amount of time previously remaining in the alarm clock of the
calling process.
NAME
brk, sbrk — change data segment space allocation

SYNOPSIS
int brk (endds)
char *endds;
char *sbrk (incr)
int incr;

DESCRIPTION
brk and sbrk are used to change dynamically the amount of space allocated for
the calling process’s data segment [see exec(2)]. The change is made by resetting
the process’s break value and allocating the appropriate amount of space. The
break value is the address of the first location beyond the end of the data seg­
ment. The amount of allocated space increases as the break value increases.
Newly allocated space is set to zero. If, however, the same memory space is
reallocated to the same process its contents are undefined.
brk sets the break value to endds and changes the allocated space accordingly.
Sbrk adds incr bytes to the break value and changes the allocated space accord­
ingly. Incr can be negative, in which case the amount of allocated space is
decreased.

brk and sbrk will fail without making any change in the allocated space if one or
more of the following are true:

[ENOMEM] Such a change would result in more space being allocated
than is allowed by the system-imposed maximum process
size [see ulimit(2)].

[EAGAIN] Total amount of system memory available for a read
during physical IO is temporarily insufficient [see
shmop(2)]. This may occur even though the space
requested was less than the system-imposed maximum
process size [see ulimit(2)].

SEE ALSO
exec(2), shmop(2), ulimit(2), end(3C).

DIAGNOSTICS
Upon successful completion, brk returns a value of 0 and sbrk returns the old
break value. Otherwise, a value of −1 is returned and errno is set to indicate the
error.
NAME
  chdir — change working directory

SYNOPSIS
  int chdir (path)
  char *path;

DESCRIPTION
  Path points to the path name of a directory. chdir causes the named directory to become the current working directory, the starting point for path searches for path names not beginning with /.
  chdir will fail and the current working directory will be unchanged if one or more of the following are true:

  [ENOTDIR] A component of the path name is not a directory.
  [ENOENT] The named directory does not exist.
  [EACCES] Search permission is denied for any component of the path name.
  [EFAULT] Path points outside the allocated address space of the process.
  [EINTR] A signal was caught during the chdir system call.
  [ENOLINK] Path points to a remote machine and the link to that machine is no longer active.
  [EMULTIHOOP] Components of path require hopping to multiple remote machines.

SEE ALSO
  chroot(2).

DIAGNOSTICS
  Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
NAME
chmod — change mode of file

SYNOPSIS
int chmod (path, mode)
char *path;
int mode;

DESCRIPTION
Path points to a path name naming a file. chmod sets the access permission portion of the named file's mode according to the bit pattern contained in mode.

Access permission bits are interpreted as follows:

<table>
<thead>
<tr>
<th>Mode Bit</th>
<th>Access Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>04000</td>
<td>Set user ID on execution.</td>
</tr>
<tr>
<td>020#0</td>
<td>Set group ID on execution if # is 7, 5, 3, or 1</td>
</tr>
<tr>
<td></td>
<td>Enable mandatory file/record locking if # is 6, 4, 2, or 0</td>
</tr>
<tr>
<td>01000</td>
<td>Save text image after execution.</td>
</tr>
<tr>
<td>00400</td>
<td>Read by owner.</td>
</tr>
<tr>
<td>00200</td>
<td>Write by owner.</td>
</tr>
<tr>
<td>00100</td>
<td>Execute (search if a directory) by owner.</td>
</tr>
<tr>
<td>00070</td>
<td>Read, write, execute (search) by group.</td>
</tr>
<tr>
<td>00007</td>
<td>Read, write, execute (search) by others.</td>
</tr>
</tbody>
</table>

The effective user ID of the process must match the owner of the file or be super-user to change the mode of a file.

If the effective user ID of the process is not super-user, mode bit 01000 (save text image on execution) is cleared.

If the effective user ID of the process is not super-user and the effective group ID of the process does not match the group ID of the file, mode bit 02000 (set group ID on execution) is cleared.

If a 410 executable file has the sticky bit (mode bit 01000) set, the operating system will not delete the program text from the swap area when the last user process terminates. If a 413 executable file has the sticky bit set, the operating system will not delete the program text from memory when the last user process terminates. In either case, if the sticky bit is set the text will already be available (either in a swap area or in memory) when the next user of the file executes it, thus making execution faster.

If the mode bit 02000 (set group ID on execution) is set and the mode bit 00010 (execute or search by group) is not set, mandatory file/record locking will exist on a regular file. This may effect future calls to open(2), creat(2), read(2), and write(2) on this file.

chmod will fail and the file mode will be unchanged if one or more of the following are true:

- [ENOTDIR] A component of the path prefix is not a directory.
- [ENOENT] The named file does not exist.
- [EACCES] Search permission is denied on a component of the path prefix.
[EPERM] The effective user ID does not match the owner of the file and the effective user ID is not super-user.

[EROFS] The named file resides on a read-only file system.

[EFAULT] Path points outside the allocated address space of the process.

[EINTR] A signal was caught during the chmod system call.

[ENOLINK] Path points to a remote machine and the link to that machine is no longer active.

[EMULTIHOP] Components of path require hopping to multiple remote machines.

SEE ALSO
chown(2), creat(2), fcntl(2), mknod(2), open(2), read(2), write(2).


DIAGNOSTICS
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.
NAME
cownik -- change owner and group of a file

SYNOPSIS
int chown (path, owner, group)
char *path;
int owner, group;

DESCRIPTION
Path points to a path name naming a file. The owner ID and group ID of the
named file are set to the numeric values contained in owner and group respec-
tively.

Only processes with effective user ID equal to the file owner or super-user may
change the ownership of a file.

If chown is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

chown will fail and the owner and group of the named file will remain unchanged if one or more of the following are true:

[ENOTDIR] A component of the path prefix is not a directory.
[ENOENT] The named file does not exist.
[EACCES] Search permission is denied on a component of the path prefix.
[EPERM] The effective user ID does not match the owner of the file and
the effective user ID is not super-user.
[EROFS] The named file resides on a read-only file system.
[EFAULT] Path points outside the allocated address space of the process.
[EINTR] A signal was caught during the chown system call.
[ENOLINK] Path points to a remote machine and the link to that machine is
no longer active.
[EMULTIHOP] Components of path require hopping to multiple remote
machines.

SEE ALSO
chmod(2).

DIAGNOSTICS
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1
is returned and errno is set to indicate the error.
NAME
chroot — change root directory

SYNOPSIS

```c
int chroot (path)
char *path;
```

DESCRIPTION
Path points to a path name naming a directory. chroot causes the named directory to become the root directory, the starting point for path searches for path names beginning with `/`. The user’s working directory is unaffected by the chroot system call.

The effective user ID of the process must be super-user to change the root directory.

The .. entry in the root directory is interpreted to mean the root directory itself. Thus, .. cannot be used to access files outside the subtree rooted at the root directory.

chroot will fail and the root directory will remain unchanged if one or more of the following are true:

- `[ENOTDIR]` Any component of the path name is not a directory.
- `[ENOENT]` The named directory does not exist.
- `[EPERM]` The effective user ID is not super-user.
- `[EFAULT]` Path points outside the allocated address space of the process.
- `[EINTR]` A signal was caught during the chroot system call.
- `[ENOLINK]` Path points to a remote machine and the link to that machine is no longer active.
- `[EMULTIHOP]` Components of path require hopping to multiple remote machines.

SEE ALSO
chdir(2).

DIAGNOSTICS
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
NAME
   close — close a file descriptor

SYNOPSIS
   int close (fildes)
   int fildes;

DESCRIPTION
   Fildes is a file descriptor obtained from a creat, open, dup, fcntl, or pipe system call. close closes the file descriptor indicated by fildes. All outstanding record locks owned by the process (on the file indicated by fildes) are removed.

   If a STREAMS [see intro(2)] file is closed, and the calling process had previously registered to receive a SIGPOLL signal [see signal(2) and sigset(2)] for events associated with that file [see I_SETSIG in streamio(7)], the calling process will be unregistered for events associated with the file. The last close for a stream causes the stream associated with fildes to be dismantled. If O_NDELAY is not set and there have been no signals posted for the stream, close waits up to 15 seconds, for each module and driver, for any output to drain before dismantling the stream. If the O_NDELAY flag is set or if there are any pending signals, close does not wait for output to drain, and dismantles the stream immediately.

   The named file is closed unless one or more of the following are true:

   [EBADF] Fildes is not a valid open file descriptor.
   [EINVAL] A signal was caught during the close system call.
   [ENOLINK] Fildes is on a remote machine and the link to that machine is no longer active.

SEE ALSO

DIAGNOSTICS
   Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
NAME
creat — create a new file or rewrite an existing one

SYNOPSIS

int creat (path, mode)
char *path;
int mode;

DESCRIPTION
creat creates a new ordinary file or prepares to rewrite an existing file named by
the path name pointed to by path.

If the file exists, the length is truncated to 0 and the mode and owner are
unchanged. Otherwise, the file’s owner ID is set to the effective user ID, of the
process the group ID of the process is set to the effective group ID, of the process
and the low-order 12 bits of the file mode are set to the value of mode modified
as follows:

All bits set in the process’s file mode creation mask are cleared [see
umask(2)].

The “save text image after execution bit” of the mode is cleared [see
chmod(2)].

Upon successful completion, a write-only file descriptor is returned and the file
is open for writing, even if the mode does not permit writing. The file pointer is
set to the beginning of the file. The file descriptor is set to remain open across
exec system calls [see fcntl(2)]. No process may have more than 20 files open
simultaneously. A new file may be created with a mode that forbids writing.
creat fails if one or more of the following are true:

[ENOTDIR] A component of the path prefix is not a directory.
[ENOENT] A component of the path prefix does not exist.
[EACCES] Search permission is denied on a component of the path prefix.
[ENOENT] The path name is null.
[EACCES] The file does not exist and the directory in which the file is to
be created does not permit writing.
[EROFS] The named file resides or would reside on a read-only file
system.
[ETXTBSY] The file is a pure procedure (shared text) file that is being exe-
cuted.
[EACCES] The file exists and write permission is denied.
[EISDIR] The named file is an existing directory.
[EMFILE] NOFILES file descriptors are currently open.
[EFAULT] Path points outside the allocated address space of the process.
[ENFILE] The system file table is full.
[EAGAIN] The file exists, mandatory file/record locking is set, and there
are outstanding record locks on the file [see chmod(2)].
[EINTR] A signal was caught during the `creat` system call.

[ENOLINK] *Path* points to a remote machine and the link to that machine is no longer active.

[EMULTIHOP] Components of *path* require hopping to multiple remote machines.

[ENOSPC] The file system is out of inodes.

SEE ALSO

`chmod(2), close(2), dup(2), fcntl(2), lseek(2), open(2), read(2), umask(2), write(2)`.

DIAGNOSTICS

Upon successful completion, a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.
NAME
dup — duplicate an open file descriptor

SYNOPSIS
int dup (fildes)
int fildes;

DESCRIPTION
Fildes is a file descriptor obtained from a creat, open, dup, fcntl, or pipe system call. dup returns a new file descriptor having the following in common with the original:

Same open file (or pipe).
Same file pointer (i.e., both file descriptors share one file pointer).
Same access mode (read, write or read/write).

The new file descriptor is set to remain open across exec system calls [see fcntl(2)].

The file descriptor returned is the lowest one available.
dup will fail if one or more of the following are true:
[EBADF] Fildes is not a valid open file descriptor.
[EINTR] A signal was caught during the dup system call.
[EMFILE] NOFILES file descriptors are currently open.
[ENOLINK] Fildes is on a remote machine and the link to that machine is no longer active.

SEE ALSO
close(2), creat(2), exec(2), fcntl(2), open(2), pipe(2), lockf(3C).

DIAGNOSTICS
Upon successful completion a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

150
NAME
exec: execl, execv, execlp, execvp — execute a file

SYNOPSIS
int execl (path, arg0, arg1, ..., argn, (char *)0)
char *path, *arg0, *arg1, ..., *argn;
int execv (path, argv)
char *path, *argv[ ];
int execlp (file, arg0, arg1, ..., argn, (char *)0)
char *file, *arg0, *arg1, ..., *argn;
int execvp (file, argv)
char *file, *argv[ ];

DESCRIPTION
exec in all its forms transforms the calling process into a new process. The new
process is constructed from an ordinary, executable file called the new process
file. This file consists of a header [see a.out(4)], a text segment, and a data seg­
ment. The data segment contains an initialized portion and an uninitialized por­
tion (bss). There can be no return from a successful exec because the calling
process is overlaid by the new process.

When a C program is executed, it is called as follows:

main (argc, argv, envp)
int argc;
char **argv, **envp;

where argc is the argument count, argv is an array of character pointers to the
arguments themselves, and envp is an array of character pointers to the environ­
ment strings. As indicated, argc is conventionally at least one and the first
member of the array points to a string containing the name of the file.
Path points to a path name that identifies the new process file.
File points to the new process file. The path prefix for this file is obtained by a
search of the directories passed as the environment line "PATH =" [see environ(5)]. The environment is supplied by the shell [see sh(1)].
Arg0, arg1, ..., argn are pointers to null-terminated character strings. These
strings constitute the argument list available to the new process. By convention,
at least arg0 must be present and point to a string that is the same as path (or its
last component).
Argv is an array of character pointers to null-terminated strings. These strings
constitute the argument list available to the new process. By convention, argv
must have at least one member, and it must point to a string that is the same as
path (or its last component). Argv is terminated by a null pointer.
Envp is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process. Envp is terminated by a null pointer. For exec1 and execv, the C run-time start-off routine places a pointer to the environment of the calling process in the global cell:

```c
extern char **environ;
```
and it is used to pass the environment of the calling process to the new process.

File descriptors open in the calling process remain open in the new process, except for those whose close-on-exec flag is set; see fcntl(2). For those file descriptors that remain open, the file pointer is unchanged.

Signals set to terminate the calling process will be set to terminate the new process. Signals set to be ignored by the calling process will be set to be ignored by the new process. Signals set to be caught by the calling process will be set to terminate new process; see signal(2).

For signals set by sigset(2), exec will ensure that the new process has the same system signal action for each signal type whose action is SIG_DFL, SIG_IGN, or SIG_HOLD as the calling process. However, if the action is to catch the signal, then the action will be reset to SIG_DFL, and any pending signal for this type will be held.

If the set-user-ID mode bit of the new process file is set [see chmod(2)], exec sets the effective user ID of the new process to the owner ID of the new process file. Similarly, if the set-group-ID mode bit of the new process file is set, the effective group ID of the new process is set to the group ID of the new process file. The real user ID and real group ID of the new process remain the same as those of the calling process.

The shared memory segments attached to the calling process will not be attached to the new process [see shmop(2)].

Profiling is disabled for the new process; see profi(2).

The new process also inherits the following attributes from the calling process:

- nice value [see nice(2)]
- process ID
- parent process ID
- process group ID
- semadj values [see semop(2)]
- tty group ID [see exit(2) and signal(2)]
- trace flag [see ptrace(2) request 0]
- time left until an alarm clock signal [see alarm(2)]
- current working directory
- root directory
- file mode creation mask [see umask(2)]
- file size limit [see ulimit(2)]
- utime, stime, cutime, and cstime [see times(2)]
- file-locks [see fcntl(2) and lockf(3C)]

exec will fail and return to the calling process if one or more of the following are true:
[ENOENT] One or more components of the new process path name of the file do not exist.
[ENOTDIR] A component of the new process path of the file prefix is not a directory.
[EACCES] Search permission is denied for a directory listed in the new process file's path prefix.
[EACCES] The new process file is not an ordinary file.
[EACCES] The new process file mode denies execution permission.
[ENOEXEC] The exec is not an execvp or execvp, and the new process file has the appropriate access permission but an invalid magic number in its header.
[ETXTBSY] The new process file is a pure procedure (shared text) file that is currently open for writing by some process.
[ENOMEM] The new process requires more memory than is allowed by the system-imposed maximum MAXMEM.
[E2BIG] The number of bytes in the new process's argument list is greater than the system-imposed limit of 5120 bytes.
[EFAULT] Required hardware is not present.
[EFAULT] An a.out that was compiled with the MAU or 32B flag is running on a machine without a MAU or 32B.
[EFAULT] Path, argv, or envp point to an illegal address.
[EAGAIN] Not enough memory.
[ELIBACC] Required shared library does not have execute permission.
[ELIBEXEC] Trying to exec(2) a shared library directly.
[EINTR] A signal was caught during the exec system call.
[ENOLINK] Path points to a remote machine and the link to that machine is no longer active.
[EMULTIHOP] Components of path require hopping to multiple remote machines.

SEE ALSO
alarm(2), exit(2), fcntl(2), fork(2), nice(2), ptrace(2), semop(2), signal(2), sigset(2), times(2), ulimit(2), umask(2), lockf(3C), a.out(4), environ(5).

DIAGNOSTICS
If exec returns to the calling process an error has occurred; the return value will be -1 and errno will be set to indicate the error.
NAME
exit, _exit — terminate process

SYNOPSIS

```c
void exit (status)
      int status;

void _exit (status)
      int status;
```

DESCRIPTION

`exit` terminates the calling process with the following consequences:

All of the file descriptors open in the calling process are closed.

If the parent process of the calling process is executing a `wait`, it is notified of
the calling process’s termination and the low order eight bits (i.e., bits 0377) of
`status` are made available to it [see `wait(2)`].

If the parent process of the calling process is not executing a `wait`, the calling
process is transformed into a zombie process. A zombie process is a process that
only occupies a slot in the process table. It has no other space allocated either
in user or kernel space. The process table slot that it occupies is partially over­
laid with time accounting information (see `<sys/proc.h>`) to be used by `times`.

The parent process ID of all of the calling processes’ existing child processes and
zombie processes is set to 1. This means the initialization process [see `intro(2)`]
inherits each of these processes.

Each attached shared memory segment is detached and the value of
`shm_nattach` in the data structure associated with its shared memory identifier is
decremented by 1.

For each semaphore for which the calling process has set a semadj value [see `semop(2)`],
that semadj value is added to the semval of the specified semaphore.

If the process has a process, text, or data lock, an `unlock` is performed [see `plock(2)`].

An accounting record is written on the accounting file if the system’s accounting
routine is enabled [see `acct(2)`].

If the process ID, tty group ID, and process group ID of the calling process are
equal, the `SIGHUP` signal is sent to each process that has a process group ID
equal to that of the calling process.

A death of child signal is sent to the parent.

The C function `exit` may cause cleanup actions before the process exits. The
function `_exit` circumvents all cleanup.
SEE ALSO
    acct(2), intro(2), plock(2), semop(2), signal(2), sigset(2), wait(2).

WARNING
    See WARNING in signal(2).

DIAGNOSTICS
    None. There can be no return from an exit system call.
NAME
fcntl – file control

SYNOPSIS
#include <fcntl.h>
int fcntl (fd, cmd, arg)
int fd, cmd, arg;

DESCRIPTION
fcntl provides for control over open files. Fd is an open file descriptor
obtained from a creat, open, dup, fcntl, or pipe system call.

The commands available are:

F_DUPFD  Return a new file descriptor as follows:
Lowest numbered available file descriptor greater than or equal
to arg.
Same open file (or pipe) as the original file.
Same file pointer as the original file (i.e., both file descriptors
share one file pointer).
Same access mode (read, write or read/write).
Same file status flags (i.e., both file descriptors share the same
file status flags).
The close-on-exec flag associated with the new file descriptor
is set to remain open across exec(2) system calls.

F_GETFD  Get the close-on-exec flag associated with the file descriptor
fd. If the low-order bit is 0 the file will remain open across
exec, otherwise the file will be closed upon execution of exec.

F_SETFD  Set the close-on-exec flag associated with fd to the low-
order bit of arg (0 or 1 as above).

F_GETFL  Get file status flags.

F_SETFL  Set file status flags to arg. Only certain flags can be set [see
fcntl(5)].

F_GETLK  Get the first lock which blocks the lock description given by
the variable of type struct flock pointed to by arg. The infor-
mation retrieved overwrites the information passed to fcntl in
the flock structure. If no lock is found that would prevent this
lock from being created, then the structure is passed back
unchanged except for the lock type which will be set to
F_UNLCK.

F_SETLK  Set or clear a file segment lock according to the variable of
type struct flock pointed to by arg [see fcntl(5)]. The cmd
F_SETLK is used to establish read (F_RDLCK) and write
(F_WRLCK) locks, as well as remove either type of lock
(F_UNLCK). If a read or write lock cannot be set fcntl will
return immediately with an error value of −1.
F_SETLK

This cmd is the same as F_SETLK except that if a read or write lock is blocked by other locks, the process will sleep until the segment is free to be locked.

A read lock prevents any process from write locking the protected area. More than one read lock may exist for a given segment of a file at a given time. The file descriptor on which a read lock is being placed must have been opened with read access.

A write lock prevents any process from read locking or write locking the protected area. Only one write lock may exist for a given segment of a file at a given time. The file descriptor on which a write lock is being placed must have been opened with write access.

The structure `flock` describes the type (`l_type`), starting offset (`l_whence`), relative offset (`l_start`), size (`l_len`), process id (`l_pid`), and RFS system id (`l_sysid`) of the segment of the file to be affected. The process id and system id fields are used only with the F_GETLK cmd to return the values for a blocking lock. Locks may start and extend beyond the current end of a file, but may not be negative relative to the beginning of the file. A lock may be set to always extend to the end of file by setting `l_len` to zero (0). If such a lock also has `l_whence` and `l_start` set to zero (0), the whole file will be locked. Changing or unlocking a segment from the middle of a larger locked segment leaves two smaller segments for either end. Locking a segment that is already locked by the calling process causes the old lock type to be removed and the new lock type to take effect. All locks associated with a file for a given process are removed when a file descriptor for that file is closed by that process or the process holding that file descriptor terminates. Locks are not inherited by a child process in a `fork(2)` system call.

When mandatory file and record locking is active on a file, [see `chmod(2)`], read and write system calls issued on the file will be affected by the record locks in effect.

`fcntl` will fail if one or more of the following are true:

- `[EBADF]` `Fildes` is not a valid open file descriptor.
- `[EINVAL]` `Cmd` is F_DUPFD. `arg` is either negative, or greater than or equal to the configured value for the maximum number of open file descriptors allowed each user.
- `[EINVAL]` `Cmd` is F_GETLK, F_SETLK, or SETLKW and `arg` or the data it points to is not valid.
- `[EACCES]` `Cmd` is F_SETLK the type of lock (`l_type`) is a read (F_RDLCK) lock and the segment of a file to be locked is already write locked by another process or the type is a write (F_WRLCK) lock and the segment of a file to be locked is already read or write locked by another process.
Cmd is F_SETLK or F_SETLKW, the type of lock is a read or write lock, and there are no more record locks available (too many file segments locked) because the system maximum has been exceeded.

Cmd is F_SETLKW, the lock is blocked by some lock from another process, and putting the calling-process to sleep, waiting for that lock to become free, would cause a deadlock.

Cmd is F_SETLK, arg points outside the program address space.

A signal was caught during the fcntl system call.

Fildes is on a remote machine and the link to that machine is no longer active.

SEE ALSO
close(2), creat(2), dup(2), exec(2), fork(2), open(2), pipe(2), fcntl(5).

DIAGNOSTICS
Upon successful completion, the value returned depends on cmd as follows:

- F_DUPFD: A new file descriptor.
- F_GETFD: Value of flag (only the low-order bit is defined).
- F_SETFD: Value other than -1.
- F_GETFL: Value of file flags.
- F_SETFL: Value other than -1.
- F_GETLK: Value other than -1.
- F_SETLK: Value other than -1.
- F_SETLKW: Value other than -1.

Otherwise, a value of -1 is returned and errno is set to indicate the error.

WARNINGS
Because in the future the variable errno will be set to EAGAIN rather than EACCES when a section of a file is already locked by another process, portable application programs should expect and test for either value.
NAME
fork — create a new process

SYNOPSIS
int fork()

DESCRIPTION
fork causes creation of a new process. The new process (child process) is an
exact copy of the calling process (parent process). This means the child process
inherits the following attributes from the parent process:

- environment
- close-on-exec flag [see exec(2)]
- signal handling settings (i.e., SIG_DFL, SIG_IGN, SIG_HOLD, function
  address)
- set-user-ID mode bit
- set-group-ID mode bit
- profiling on/off status
- nice value [see nice(2)]
- all attached shared memory segments [see shmem(2)]
- process group ID
- tty group ID [see exit(2)]
- current working directory
- root directory
- file mode creation mask [see umask(2)]
- file size limit [see ulimit(2)]

The child process differs from the parent process in the following ways:

- The child process has a unique process ID.
- The child process has a different parent process ID (i.e., the process ID of
  the parent process).
- The child process has its own copy of the parent’s file descriptors. Each
  of the child’s file descriptors shares a common file pointer with the
  corresponding file descriptor of the parent.
- All semadj values are cleared [see semop(2)].
- Process locks, text locks and data locks are not inherited by the child
  [see plock(2)].
- The child process’s utime, stime, cutime, and cstime are set to 0. The
  time left until an alarm clock signal is reset to 0.

fork will fail and no child process will be created if one or more of the following
are true:

[EAGAIN] The system-imposed limit on the total number of processes
under execution would be exceeded.

[EAGAIN] The system-imposed limit on the total number of processes
under execution by a single user would be exceeded.

[EAGAIN] Total amount of system memory available when reading via
raw IO is temporarily insufficient.
SEE ALSO
exec(2), nice(2), plock(2), ptrace(2), semop(2), shmop(2), signal(2), sigset(2),
times(2), ulimit(2), umask(2), wait(2).

DIAGNOSTICS
Upon successful completion, fork returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of -1 is returned to the parent process, no child process is created, and errno is set to indicate the error.
NAME
getdents — read directory entries and put in a file system independent format

SYNOPSIS
#include <sys/dirent.h>
int getdents (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;

DESCRIPTION
Fildes is a file descriptor obtained from an open(2) or dup(2) system call.
getdents attempts to read nbyte bytes from the directory associated with fildes
and to format them as file system independent directory entries in the buffer
pointed to by buf. Since the file system independent directory entries are of
variable length, in most cases the actual number of bytes returned will be strictly
less than nbyte.
The file system independent directory entry is specified by the dirent structure.
For a description of this see dirent(4).
On devices capable of seeking, getdents starts at a position in the file given by
the file pointer associated with fildes. Upon return from getdents, the file pointer
is incremented to point to the next directory entry.
This system call was developed in order to implement the readdir(3X) routine
[for a description see directory(3X)], and should not be used for other purposes.
getdents will fail if one or more of the following are true:
[EBADF]  Fildes is not a valid file descriptor open for reading.
[EFAULT] Buf points outside the allocated address space.
[EINVAL]  nbyte is not large enough for one directory entry.
[ENOENT]  The current file pointer for the directory is not located at a valid
entry.
[ENOLINK]  Fildes points to a remote machine and the link to that machine
is no longer active.
[ENOTDIR]  Fildes is not a directory.
[EIO]  An I/O error occurred while accessing the file system.

SEE ALSO
directory(3X), dirent(4).

DIAGNOSTICS
Upon successful completion a non-negative integer is returned indicating the
number of bytes actually read. A value of 0 indicates the end of the directory
has been reached. If the system call failed, a -1 is returned and errno is set to
indicate the error.
NAME

getmsg — get next message off a stream

SYNOPSIS

#include <stropts.h>

int getmsg(fd, ctlptr, dataptr, flags)
int fd;
struct strbuf *ctlptr;
struct strbuf *dataptr;
int *flags;

DESCRIPTION

getmsg retrieves the contents of a message [see intro(2)] located at the stream head read queue from a STREAMS file, and places the contents into user specified buffer(s). The message must contain either a data part, a control part or both. The data and control parts of the message are placed into separate buffers, as described below. The semantics of each part is defined by the STREAMS module that generated the message.

Fd specifies a file descriptor referencing an open stream. Ctlptr and dataptr each point to a strbuf structure which contains the following members:

    int maxlen; /* maximum buffer length */
    int len;   /* length of data */
    char *buf; /* ptr to buffer */

where buf points to a buffer in which the data or control information is to be placed, and maxlen indicates the maximum number of bytes this buffer can hold. On return, len contains the number of bytes of data or control information actually received, or is 0 if there is a zero-length control or data part, or is -1 if no data or control information is present in the message. Flags may be set to the values 0 or RS_HIPRI and is used as described below.

Ctlptr is used to hold the control part from the message and dataptr is used to hold the data part from the message. If ctlptr (or dataptr) is NULL or the maxlen field is -1, the control (or data) part of the message is not processed and is left on the stream head read queue and len is set to -1. If the maxlen field is set to 0 and there is a zero-length control (or data) part, that zero-length part is removed from the read queue and len is set to 0. If the maxlen field is set to 0 and there are more than zero bytes of control (or data) information, that information is left on the read queue and len is set to 0. If the maxlen field in ctlptr or dataptr is less than, respectively, the control or data part of the message, maxlen bytes are retrieved. In this case, the remainder of the message is left on the stream head read queue and a non-zero return value is provided, as described below under DIAGNOSTICS. If information is retrieved from a priority message, flags is set to RS_HIPRI on return.

By default, getmsg processes the first priority or non-priority message available on the stream head read queue. However, a user may choose to retrieve only priority messages by setting flags to RS_HIPRI. In this case, getmsg will only process the next message if it is a priority message.

If O_NDELAY has not been set, getmsg blocks until a message, of the type(s) specified by flags (priority or either), is available on the stream head read queue.
If O_NDELAY has been set and a message of the specified type(s) is not present on the read queue, `getmsg` fails and sets `errno` to `EAGAIN`.

If a hangup occurs on the stream from which messages are to be retrieved, `getmsg` will continue to operate normally, as described above, until the stream head read queue is empty. Thereafter, it will return 0 in the `len` fields of `ctlptr` and `dataptr`.

`getmsg` fails if one or more of the following are true:

- `[EAGAIN]` The O_NDELAY flag is set, and no messages are available.
- `[EBADF]` `Fd` is not a valid file descriptor open for reading.
- `[EBADMSG]` Queued message to be read is not valid for `getmsg`.
- `[EFAULT]` `Ctlptr`, `dataptr`, or `flags` points to a location outside the allocated address space.
- `[EINTR]` A signal was caught during the `getmsg` system call.
- `[EINVAL]` An illegal value was specified in `flags`, or the stream referenced by `fd` is linked under a multiplexor.
- `[ENOSTR]` A stream is not associated with `fd`.

A `getmsg` can also fail if a STREAMS error message had been received at the stream head before the call to `getmsg`. The error returned is the value contained in the STREAMS error message.

SEE ALSO

`intro(2), read(2), poll(2), putmsg(2), write(2)`.

STREAMS Primer
STREAMS Programmer's Guide

DIAGNOSTICS

Upon successful completion, a non-negative value is returned. A value of 0 indicates that a full message was read successfully. A return value of `MORECTL` indicates that more control information is waiting for retrieval. A return value of `MOREDATA` indicates that more data is waiting for retrieval. A return value of `MORECT4MOREDATA` indicates that both types of information remain. Subsequent `getmsg` calls will retrieve the remainder of the message.
NAME
getpid, getpgrp, getppid — get process, process group, and parent process IDs

SYNOPSIS
int getpid()
int getpgrp()
int getppid()

DESCRIPTION
getpid returns the process ID of the calling process.
Getpgrp returns the process group ID of the calling process.
Getppid returns the parent process ID of the calling process.

SEE ALSO
exec(2), fork(2), intro(2), setpgrp(2), signal(2).
NAME  
getuid, geteuid, getgid, getegid — get real user, effective user, real group, and effective group IDs

SYNOPSIS  
unsigned short getuid ()
unsigned short geteuid ()
unsigned short getgid ()
unsigned short getegid ()

DESCRIPTION  
getuid returns the real user ID of the calling process.
Geteuid returns the effective user ID of the calling process.
Getgid returns the real group ID of the calling process.
Getegid returns the effective group ID of the calling process.

SEE ALSO  
intro(2), setuid(2).
NAME
ioctl - control device

SYNOPSIS
int ioctl (fildes, request, arg)
int fildes, request;

DESCRIPTION
ioctl performs a variety of control functions on devices and STREAMS. For non-
STREAMS files, the functions performed by this call are device-specific control
functions. The arguments request and arg are passed to the file designated by
fildes and are interpreted by the device driver. This control is infrequently used
on non-STREAMS devices, with the basic input/output functions performed
through the read(2) and write(2) system calls.

For STREAMS files, specific functions are performed by the ioctl call as described
in streamio(7).

Fildes is an open file descriptor that refers to a device. Request selects the con­trol
function to be performed and will depend on the device being addressed. Arg represents additional information that is needed by this specific device to
perform the requested function. The data type of arg depends upon the partic­ular control request, but it is either an integer or a pointer to a device-specific
data structure.

In addition to device-specific and STREAMS functions, generic functions are pro­vided by more than one device driver, for example, the general terminal inter­face [see termio(7)].

ioctl will fail for any type of file if one or more of the following are true:

[EBADF] Fildes is not a valid open file descriptor.
[ENOTTY] Fildes is not associated with a device driver that accepts control
functions.
[EINTR] A signal was caught during the ioctl system call.

ioctl will also fail if the device driver detects an error. In this case, the error is
passed through ioctl without change to the caller. A particular driver might not
have all of the following error cases. Other requests to device drivers will fail if
one or more of the following are true:

[EFAULT] Request requires a data transfer to or from a buffer pointed to by arg, but some part of the buffer is outside the process’s allo­cated space.

[EINVAL] Request or arg is not valid for this device.
[EIO] Some physical I/O error has occurred.
[ENXIO] The request and arg are valid for this device driver, but the ser­vice requested can not be performed on this particular sub­device.
[ENOLINK]  *Fildes* is on a remote machine and the link to that machine is no longer active.

STREAMS errors are described in *streamio(7)*.

SEE ALSO

*streamio(7)*, *termio(7)* in the *System Administrator's Reference Manual*.

DIAGNOSTICS

Upon successful completion, the value returned depends upon the device control function, but must be a non-negative integer. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.
NAME
  kill — send a signal to a process or a group of processes

SYNOPSIS
  int kill (pid, sig)
  int pid, sig;

DESCRIPTION
  kill sends a signal to a process or a group of processes. The process or group of
  processes to which the signal is to be sent is specified by pid. The signal that is
  to be sent is specified by sig and is either one from the list given in signal(2), or
  0. If sig is 0 (the null signal), error checking is performed but no signal is actu­
  ally sent. This can be used to check the validity of pid.

  The real or effective user ID of the sending process must match the real or
  effective user ID of the receiving process, unless the effective user ID of the
  sending process is super-user.

  The processes with a process ID of 0 and a process ID of 1 are special processes
  [see intro(2)] and will be referred to below as proc0 and proc1, respectively.

  If pid is greater than zero, sig will be sent to the process whose process ID is
  equal to pid. Pid may equal 1.

  If pid is 0, sig will be sent to all processes excluding proc0 and proc1 whose pro­
  cess group ID is equal to the process group ID of the sender.

  If pid is −1 and the effective user ID of the sender is not super-user, sig will be
  sent to all processes excluding proc0 and proc1 whose real user ID is equal to the
  effective user ID of the sender.

  If pid is −1 and the effective user ID of the sender is super-user, sig will be sent
  to all processes excluding proc0 and proc1.

  If pid is negative but not −1, sig will be sent to all processes whose process
  group ID is equal to the absolute value of pid.

  kill will fail and no signal will be sent if one or more of the following are true:

  [EINVAL]    Sig is not a valid signal number.
  [EINVAL]    Sig is SICKILL and pid is 1 (proc1).
  [ESRCH]     No process can be found corresponding to that specified by pid.
  [EPERM]     The user ID of the sending process is not super-user, and its real
              or effective user ID does not match the real or effective user ID
              of the receiving process.

SEE ALSO
  getpid(2), setpgroup(2), signal(2), sigset(2).

DIAGNOSTICS
  Upon successful completion, a value of 0 is returned. Otherwise, a value of −1
  is returned and errno is set to indicate the error.
NAME
   link — link to a file

SYNOPSIS
   int link (path1, path2)
   char *path1, *path2;

DESCRIPTION
   Path1 points to a path name naming an existing file. Path2 points to a path
   name naming the new directory entry to be created. link creates a new link
   (directory entry) for the existing file.

   link will fail and no link will be created if one or more of the following are true:

   [ENOTDIR] A component of either path prefix is not a directory.
   [ENOENT] A component of either path prefix does not exist.
   [EACCES] A component of either path prefix denies search permission.
   [ENOENT] The file named by path1 does not exist.
   [EEXIST] The link named by path2 exists.
   [EPERM] The file named by path1 is a directory and the effective user ID
          is not super-user.
   [EXDEV] The link named by path2 and the file named by path1 are on
          different logical devices (file systems).
   [ENOENT] Path2 points to a null path name.
   [EACCES] The requested link requires writing in a directory with a mode
          that denies write permission.
   [EROFS] The requested link requires writing in a directory on a read-only
          file system.
   [EFAULT] Path points outside the allocated address space of the process.
   [EMLINK] The maximum number of links to a file would be exceeded.
   [EINTR] A signal was caught during the link system call.
   [ENOLINK] Path points to a remote machine and the link to that machine is
          no longer active.
   [EMULTIHOP] Components of path require hopping to multiple remote
          machines.

SEE ALSO
   unlink(2).

DIAGNOSTICS
   Upon successful completion, a value of 0 is returned. Otherwise, a value of −1
   is returned and errno is set to indicate the error.
NAME
lseek — move read/write file pointer

SYNOPSIS
long lseek (fildes, offset, whence)
int fildes;
long offset;
int whence;

DESCRIPTION
"Fildes is a file descriptor returned from a creat, open, dup, or fcntl system call. lseek sets the file pointer associated with fildes as follows:
   If whence is 0, the pointer is set to offset bytes.
   If whence is 1, the pointer is set to its current location plus offset.
   If whence is 2, the pointer is set to the size of the file plus offset.

Upon successful completion, the resulting pointer location, as measured in bytes from the beginning of the file, is returned. Note that if fildes is a remote file descriptor and offset is negative, lseek will return the file pointer even if it is negative.

lseek will fail and the file pointer will remain unchanged if one or more of the following are true:

[EBADF] Fildes is not an open file descriptor.
[ESPIPE] Fildes is associated with a pipe or fifo.
[EINVAL and SIGSYS signal]
   Whence is not 0, 1, or 2.
[EINVAL] Fildes is not a remote file descriptor, and the resulting file pointer would be negative.

Some devices are incapable of seeking. The value of the file pointer associated with such a device is undefined.

SEE ALSO
creat(2), dup(2), fcntl(2), open(2).

DIAGNOSTICS
Upon successful completion, a non-negative integer indicating the file pointer value is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
NAME
mdir — make a directory

SYNOPSIS

int mkdir (path, mode)
char *path;
int mode;

DESCRIPTION
The routine mkdir creates a new directory with the name path. The mode of the
new directory is initialized from the mode. The protection part of the mode argument is modified by
the process’s mode mask [see umask(2)].

The directory’s owner ID is set to the process’s effective user ID. The directory’s
group ID is set to the process’s effective group ID. The newly created directory is empty with the possible exception
of entries for "." and "..". mkdir will fail and no directory will be created if one or more of the following are true:

[ENOTDIR] A component of the path prefix is not a directory.
[ENOENT] A component of the path prefix does not exist.
[ENOLINK] Path points to a remote machine and the link to that machine is no longer active.
[EMULTIHOP] Components of path require hopping to multiple remote machines.
[EACCES] Either a component of the path prefix denies search permission or write permission is denied on the parent directory of the
directory to be created.
[ENOENT] The path is longer than the maximum allowed.
[EEXIST] The named file already exists.
[EROFS] The path prefix resides on a read-only file system.
[EFAULT] Path points outside the allocated address space of the process.
[EMLINK] The maximum number of links to the parent directory would be exceeded.
[EIO] An I/O error has occurred while accessing the file system.

DIAGNOSTICS
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and errno is set to indicate the error.
NAME
mknod — make a directory, or a special or ordinary file

SYNOPSIS

```c
int mknod (path, mode, dev)
char *path;
int mode, dev;
```

DESCRIPTION

`mknod` creates a new file named by the path name pointed to by `path`. The mode of the new file is initialized from `mode`. Where the value of `mode` is interpreted as follows:

- **0170000** file type; one of the following:
  - **0010000** fifo special
  - **0020000** character special
  - **0040000** directory
  - **0060000** block special
  - **0100000** or **0000000** ordinary file
- **0004000** set user ID on execution
- **00020#0** set group ID on execution if # is 7, 5, 3, or 1
  - enable mandatory file/record locking if # is 6, 4, 2, or 0
- **0001000** save text image after execution
- **0000777** access permissions; constructed from the following:
  - **0000400** read by owner
  - **0000200** write by owner
  - **0000100** execute (search on directory) by owner
  - **0000070** read, write, execute (search) by group
  - **0000007** read, write, execute (search) by others

The owner ID of the file is set to the effective user ID of the process. The group ID of the file is set to the effective group ID of the process.

Values of `mode` other than those above are undefined and should not be used. The low-order 9 bits of `mode` are modified by the process's file mode creation mask: all bits set in the process's file mode creation mask are cleared [see `umask(2)`]. If `mode` indicates a block or character special file, `dev` is a configuration-dependent specification of a character or block I/O device. If `mode` does not indicate a block special or character special device, `dev` is ignored.

`mknod` may be invoked only by the super-user for file types other than FIFO special.

`mknod` will fail and the new file will not be created if one or more of the following are true:

- **[EPERM]** The effective user ID of the process is not super-user.
- **[ENOTDIR]** A component of the path prefix is not a directory.
- **[ENOENT]** A component of the path prefix does not exist.
The directory in which the file is to be created is located on a read-only file system.
The named file exists.
Path points outside the allocated address space of the process.
No space is available.
A signal was caught during the mknod system call.
Path points to a remote machine and the link to that machine is no longer active.
Components of path require hopping to multiple remote machines.

SEE ALSO
chmod(2), exec(2), umask(2), fs(4).

DIAGNOSTICS
Upon successful completion a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

WARNING
If mknod is used to create a device in a remote directory (Remote File Sharing), the major and minor device numbers are interpreted by the server.
NAME
mount — mount a file system

SYNOPSIS
#include <sys/mount.h>
int mount (spec, dir, mflag, fstyp)
char *spec, *dir;
int mflag, fstyp;

DESCRIPTION
mount requests that a removable file system contained on the block special file
identified by spec be mounted on the directory identified by dir. Spec and dir
are pointers to path names. Fstyp is the file system type number. The sysfs(2)
sequence can be used to determine the file system type number. Note that if
the MS_FSS flag bit of mflag is off, the file system type will default to the root
file system type. Only if the bit is on will fstyp be used to indicate the file
system type.

Upon successful completion, references to the file dir will refer to the root direc-
tory on the mounted file system.

The low-order bit of mflag is used to control write permission on the mounted
file system; if 1, writing is forbidden, otherwise writing is permitted according to
individual file accessibility.

mount may be invoked only by the super-user. It is intended for use only by
the mount(1M) utility.

mount will fail if one or more of the following are true:

[EPERM] The effective user ID is not super-user.
[ENOENT] Any of the named files does not exist.
[ENOTDIR] A component of a path prefix is not a directory.
[EREMOTE] Spec is remote and cannot be mounted.
[ENOLINK] Path points to a remote machine and the link to that machine is
no longer active.
[EMULTIHOP] Components of path require hopping to multiple remote
machines.
[ENOTBLK] Spec is not a block special device.
[ENXIO] The device associated with spec does not exist.
[ENOTDIR] Dir is not a directory.
[EFAULT] Spec or dir points outside the allocated address space of the
process.
[EBUSY] Dir is currently mounted on, is someone’s current working
directory, or is otherwise busy.
[EBUSY] The device associated with spec is currently mounted.
[EBUSY] There are no more mount table entries.
[EROFs] Spec is write protected and mflag requests write permission.
[ENOSSPC] The file system state in the super-block is not FsOKAY and mflag requests write permission.
[EINVAL] The super block has an invalid magic number or the fstyp is invalid or mflag is not valid.

SEE ALSO
sysfs(2), umount(2), fs(4).

DIAGNOSTICS
Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
NAME
msgctl — message control operations

SYNOPSIS
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl (msqid, cmd, buf)
int msqid, cmd;
struct msqid_ds *buf;

DESCRIPTION
msgctl provides a variety of message control operations as specified by cmd. The
following cmds are available:

IPC_STAT Place the current value of each member of the data structure
associated with msqid into the structure pointed to by buf. The
contents of this structure are defined in intro(2). {READ}

IPC_SET Set the value of the following members of the data structure
associated with msqid to the corresponding value found in the
structure pointed to by buf:
msg_perm.uid
msg_perm_gid
msg_perm.mode /* only low 9 bits */
msg_qbytes

This cmd can only be executed by a process that has an effective
user ID equal to either that of super user, or to the value of
msg_perm.cuid or msg_perm.uid in the data structure associated with msqid. Only super user can raise the value of
msg_qbytes.

IPC_RMID Remove the message queue identifier specified by msqid from
the system and destroy the message queue and data structure
associated with it. This cmd can only be executed by a process
that has an effective user ID equal to either that of super user,
or to the value of msg_perm.cuid or msg_perm.uid in the data
structure associated with msqid.

msgctl will fail if one or more of the following are true:

EINVAL Msqid is not a valid message queue identifier.
EINVAL Cmd is not a valid command.
EACCES Cmd is equal to IPC_STAT and (READ) operation permission is
denied to the calling process [see intro(2)].
EPERM Cmd is equal to IPC_RMID or IPC_SET. The effective user ID of
the calling process is not equal to that of super user, or to the
value of msg_perm.cuid or msg_perm.uid in the data structure
associated with msqid.
[EPERM]  *Cmd* is equal to **IPC_SET**, an attempt is being made to increase to the value of **msg_qbytes**, and the effective user ID of the calling process is not equal to that of super user.

[EFAULT]  *Buf* points to an illegal address.

**SEE ALSO**

intro(2), msgget(2), msgop(2).

**DIAGNOSTICS**

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.
NAME

msgget - get message queue

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget (key, msgflg)
key_t key;
int msgflg;

DESCRIPTION

msgget returns the message queue identifier associated with key.

A message queue identifier and associated message queue and data structure [see
intro(2)] are created for key if one of the following are true:

Key is equal to IPC_PRIVATE.

Key does not already have a message queue identifier associated with it,
and (msgflg & IPC_CREAT) is "true".

Upon creation, the data structure associated with the new message queue
identifier is initialized as follows:

Msg_perm.cuid, msg_perm.uid, msg_perm.cgid, and msg_perm.gid are
set equal to the effective user ID and effective group ID, respectively, of
the calling process.

The low-order 9 bits of msg_perm.mode are set equal to the low-order 9
bits of msgflg.

Msg_qnum, msg_lspid, msg_lrpid, msg_stime, and msg_rtime are set
equal to 0.

Msg_ctime is set equal to the current time.

Msg_qbytes is set equal to the system limit.

msgget will fail if one or more of the following are true:

[EACCES] A message queue identifier exists for key, but operation permi-

sion [see intro(2)] as specified by the low-order 9 bits of msgflg

would not be granted.

[ENOENT] A message queue identifier does not exist for key and (msgflg &

IPC_CREAT) is "false".

[ENOSPC] A message queue identifier is to be created but the system-

imposed limit on the maximum number of allowed message

queue identifiers system wide would be exceeded.

[EEXIST] A message queue identifier exists for key but ((msgflg &

IPC_CREAT) & (msgflg & IPC_EXCL)) is "true".

SEE ALSO

intro(2), msgctl(2), msgop(2).
DIAGNOSTICS

Upon successful completion, a non-negative integer, namely a message queue identifier, is returned. Otherwise, a value of $-1$ is returned and \textit{errno} is set to indicate the error.
NAME
    msgop — message operations

SYNOPSIS
    #include <sys/types.h>
    #include <sys/ipc.h>
    #include <sys/msg.h>

    int msgsnd (msqid, msgp, msgsz, msgflg)
    int msqid;
    struct msgbuf *msgp;
    int msgsz, msgflg;

    int msgrcv (msqid, msgp, msgsz, msgtyp, msgflg)
    int msqid;
    struct msgbuf *msgp;
    int msgsz;
    long msgtyp;
    int msgflg;

DESCRIPTION
    Msgsnd is used to send a message to the queue associated with the message
    queue identifier specified by msqid. (WRITE) Msgp points to a structure con­
    taining the message. This structure is composed of the following members:

        long mtype; /* message type */
        char mtext[]; /* message text */

    Mtype is a positive integer that can be used by the receiving process for message
    selection (see msgrcv below). Mtext is any text of length msgsz bytes. Msgsz can
    range from 0 to a system-imposed maximum.

    Msgflg specifies the action to be taken if one or more of the following are true:

        The number of bytes already on the queue is equal to msg_qbytes [see
        intro(2)].

        The total number of messages on all queues system-wide is equal to the
        system-imposed limit.

    These actions are as follows:

        If (msgflg & IPC_NOWAIT) is "true", the message will not be sent and
        the calling process will return immediately.

        If (msgflg & IPC_NOWAIT) is "false", the calling process will suspend
        execution until one of the following occurs:

        The condition responsible for the suspension no longer exists,
        in which case the message is sent.

        Msqid is removed from the system [see msgctl(2)]. When this
        occurs, errno is set equal to EIDRM, and a value of −1 is
        returned.

        The calling process receives a signal that is to be caught. In
        this case the message is not sent and the calling process
        resumes execution in the manner prescribed in signal(2).
**Msgsnd** will fail and no message will be sent if one or more of the following are true:

- **EINVAL**: *Msqid* is not a valid message queue identifier.
- **EACCES**: Operation permission is denied to the calling process [see intro(2)].
- **EINVAL**: *Mtype* is less than 1.
- **EAGAIN**: The message cannot be sent for one of the reasons cited above and (*msgflg & IPC_NOWAIT*) is "true".
- **EINVAL**: *Msgsz* is less than zero or greater than the system-imposed limit.
- **EFAULT**: *Msgp* points to an illegal address.

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* [see intro(2)].

- **Msg_qnum** is incremented by 1.
- **Msg_ispid** is set equal to the process ID of the calling process.
- **Msg_stime** is set equal to the current time.

**Msgrcv** reads a message from the queue associated with the message queue identifier specified by *msqid* and places it in the structure pointed to by *msgp*.

{READ} This structure is composed of the following members:

```c
long mtype; /* message type */
char mtext[]; /* message text */
```

*Mtype* is the received message's type as specified by the sending process. *Mtext* is the text of the message. *Msgsz* specifies the size in bytes of *mtext*. The received message is truncated to *msgsz* bytes if it is larger than *msgsz* and (*msgflg & MSG_NOERROR*) is "true". The truncated part of the message is lost and no indication of the truncation is given to the calling process.

**Msgtyp** specifies the type of message requested as follows:

- If *msgtyp* is equal to 0, the first message on the queue is received.
- If *msgtyp* is greater than 0, the first message of type *msgtyp* is received.
- If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* is received.

**Msgflg** specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

- If (*msgflg & IPC_NOWAIT*) is "true", the calling process will return immediately with a return value of −1 and *errno* set to ENOMSG.
- If (*msgflg & IPC_NOWAIT*) is "false", the calling process will suspend execution until one of the following occurs:
  
  - A message of the desired type is placed on the queue.
  - *Msqid* is removed from the system. When this occurs, *errno* is set equal to EIDRM, and a value of −1 is returned.
The calling process receives a signal that is to be caught. In this case a message is not received and the calling process resumes execution in the manner prescribed in \textit{signal(2)}.

\textit{Msgrcv} will fail and no message will be received if one or more of the following are true:

- \texttt{EINVAL} \hspace{1em} \textit{Msqid} is not a valid message queue identifier.
- \texttt{EACCES} \hspace{1em} Operation permission is denied to the calling process.
- \texttt{EINVAL} \hspace{1em} \textit{Msgsz} is less than 0.
- \texttt{E2BIG} \hspace{1em} \textit{Mtext} is greater than \textit{msgsz} and \texttt{(msgflg \& MSG\_NOERROR)} is "false".
- \texttt{ENOMEM} \hspace{1em} The queue does not contain a message of the desired type and \texttt{(msgtyp \& IPC\_NOWAIT)} is "true".
- \texttt{EFAULT} \hspace{1em} \textit{Msgp} points to an illegal address.

Upon successful completion, the following actions are taken with respect to the data structure associated with \textit{msqid} [see intro (2)].

- \textbf{Msg\_qnum} is decremented by 1.
- \textbf{Msg\_lpid} is set equal to the process ID of the calling process.
- \textbf{Msg\_rtime} is set equal to the current time.

\textbf{SEE ALSO}

intro(2), msgctl(2), msgget(2), signal(2).

\textbf{DIAGNOSTICS}

If \textit{msgsnd} or \textit{msgrcv} return due to the receipt of a signal, a value of \texttt{-1} is returned to the calling process and \textit{errno} is set to EINTR. If they return due to removal of \textit{msqid} from the system, a value of \texttt{-1} is returned and \textit{errno} is set to EIDRM.

Upon successful completion, the return value is as follows:

- \textit{Msqsnd} returns a value of 0.
- \textit{Msgrcv} returns a value equal to the number of bytes actually placed into \textit{mtext}.

Otherwise, a value of \texttt{-1} is returned and \textit{errno} is set to indicate the error.
NAME
nice — change priority of a process

SYNOPSIS
int nice (incr)
int incr;

DESCRIPTION
nice adds the value of incr to the nice value of the calling process. A process's
nice value is a non-negative number for which a more positive value results in
lower CPU priority.

A maximum nice value of 39 and a minimum nice value of 0 are imposed by the
system. (The default nice value is 20.) Requests for values above or below these
limits result in the nice value being set to the corresponding limit.

[EPERM] nice will fail and not change the nice value if incr is negative or
greater than 39 and the effective user ID of the calling process is
not super-user.

SEE ALSO
exec(2).

DIAGNOSTICS
Upon successful completion, nice returns the new nice value minus 20. Other­
wise, a value of -1 is returned and errno is set to indicate the error.
OPEN(2)

NAME
open — open for reading or writing

SYNOPSIS
#include <fcntl.h>
int open (path, oflag [, mode] )
char *path;
int oflag, mode;

DESCRIPTION
Path points to a path name naming a file. open opens a file descriptor for the
named file and sets the file status flags according to the value of oflag. For non-
STREAMS [see intro(2)] files, oflag values are constructed by or-ing flags from the
following list (only one of the first three flags below may be used):

O_RDONLY Open for reading only.
O_WRONLY Open for writing only.
O_RDWR Open for reading and writing.
O_NDELAY This flag may affect subsequent reads and writes [see read(2) and
write(2)].

When opening a FIFO with O_RDONLY or O_WRONLY set:
If O_NDELAY is set:

An open for reading-only will return without delay. An
open for writing-only will return an error if no process
currently has the file open for reading.

If O_NDELAY is clear:

An open for reading-only will block until a process opens
the file for writing. An open for writing-only will block
until a process opens the file for reading.

When opening a file associated with a communication line:
If O_NDELAY is set:

The open will return without waiting for carrier.
If O_NDELAY is clear:

The open will block until carrier is present.

O_APPEND If set, the file pointer will be set to the end of the file prior to
each write.

O_SYNC When opening a regular file, this flag affects subsequent writes.
If set, each write(2) will wait for both the file data and file status
to be physically updated.

O_CREAT If the file exists, this flag has no effect. Otherwise, the owner ID
of the file is set to the effective user ID of the process, the group
ID of the file is set to the effective group ID of the process, and
the low-order 12 bits of the file mode are set to the value of
mode modified as follows [see creat(2)]:
All bits set in the file mode creation mask of the process are cleared [see `umask(2)`].

The "save text image after execution bit" of the mode is cleared [see `chmod(2)`].

**O_TRUNC** If the file exists, its length is truncated to 0 and the mode and owner are unchanged.

**O_EXCL** If O_EXCL and O_CREAT are set, `open` will fail if the file exists.

When opening a STREAMS file, `oflag` may be constructed from O_NDELAY or-ed with either O_RDONLY, O_WRONLY or O_RDWR. Other flag values are not applicable to STREAMS devices and have no effect on them. The value of O_NDELAY affects the operation of STREAMS drivers and certain system calls [see `read(2)`, `getmsg(2)`, `putmsg(2)` and `write(2)`]. For drivers, the implementation of O_NDELAY is device-specific. Each STREAMS device driver may treat this option differently.

Certain flag values can be set following `open` as described in `fcntl(2)`.

The file pointer used to mark the current position within the file is set to the beginning of the file.

The new file descriptor is set to remain open across `exec` system calls [see `fcntl(2)`].

The named file is opened unless one or more of the following are true:

- `[EACCES]` A component of the path prefix denies search permission.
- `[EACCES]` `oflag` permission is denied for the named file.
- `[EAGAIN]` The file exists, mandatory file/record locking is set, and there are outstanding record locks on the file [see `chmod(2)`].
- `[EEXIST]` O_CREAT and O_EXCL are set, and the named file exists.
- `[EFAULT]` `Path` points outside the allocated address space of the process.
- `[EINVAL]` A signal was caught during the `open` system call.
- `[EIO]` A hangup or error occurred during a STREAMS `open`.
- `[EISDIR]` The named file is a directory and `oflag` is write or read/write.
- `[ENOMEM]` NOFILES file descriptors are currently open.
- `[EMULTIHOP]` Components of `path` require hopping to multiple remote machines.
- `[ENFILE]` The system file table is full.
- `[ENOENT]` O_CREAT is not set and the named file does not exist.
- `[ENOLINK]` `Path` points to a remote machine, and the link to that machine is no longer active.
- `[ENOMEM]` The system is unable to allocate a send descriptor.
- `[ENOSPC]` O_CREAT and O_EXCL are set, and the file system is out of inodes.
- `[ENOSR]` Unable to allocate a stream.
OPEN(2)

A component of the path prefix is not a directory.

The named file is a character special or block special file, and
the device associated with this special file does not exist.

O_NDELAY is set, the named file is a FIFO, O_WRONLY is set, and
no process has the file open for reading.

A STREAMS module or driver open routine failed.

The named file resides on a read-only file system and oflag is
write or read/write.

The file is a pure procedure (shared text) file that is being exe-
cuted and oflag is write or read/write.

A signal was caught during the open system call.

The system is unable to allocate a send descriptor.

Path points to a remote machine, and the link to that machine
is no longer active.

Components of path require hopping to multiple remote
machines.

SEE ALSO
chmod(2), close(2), creat(2), dup(2), fcntl(2), intro(2), lseek(2), read(2), getmsg(2),
putmsg(2), umask(2), write(2).

DIAGNOSTICS
Upon successful completion, the file descriptor is returned. Otherwise, a value
of -1 is returned and errno is set to indicate the error.
NAME
  pause — suspend process until signal

SYNOPSIS
  pause ()

DESCRIPTION
  pause suspends the calling process until it receives a signal. The signal must be
  one that is not currently set to be ignored by the calling process.

  If the signal causes termination of the calling process, pause will not return.

  If the signal is caught by the calling process and control is returned from the
  signal-catching function [see signal(2)], the calling process resumes execution
  from the point of suspension; with a return value of −1 from pause and errno set
to EINTR.

SEE ALSO
  alarm(2), kill(2), signal(2), sigpause(2), wait(2).
NAME
pipe — create an interprocess channel

SYNOPSIS
int pipe (fildes)
int fildes[2];

DESCRIPTION
pipe creates an I/O mechanism called a pipe and returns two file descriptors, fildes[0] and fildes[1]. Fildes[0] is opened for reading and fildes[1] is opened for writing.

Up to 5120 bytes of data are buffered by the pipe before the writing process is blocked. A read only file descriptor fildes[0] accesses the data written to fildes[1] on a first-in-first-out (FIFO) basis.

pipe will fail if:

[EMFILE] NOFILES file descriptors are currently open.
[ENFILE] The system file table is full.

SEE ALSO
read(2), write(2).

DIAGNOSTICS
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
NAME
plock — lock process, text, or data in memory

SYNOPSIS
#include <sys/lock.h>
int plock (op)
int op;

DESCRIPTION
plock allows the calling process to lock its text segment (text lock), its data segment (data lock), or both its text and data segments (process lock) into memory. Locked segments are immune to all routine swapping. plock also allows these segments to be unlocked. The effective user ID of the calling process must be super-user to use this call. Op specifies the following:

PROCLOCK — lock text and data segments into memory (process lock)
TXTLOCK — lock text segment into memory (text lock)
DATLOCK — lock data segment into memory (data lock)
UNLOCK — remove locks

plock will fail and not perform the requested operation if one or more of the following are true:

[EPERM] The effective user ID of the calling process is not super-user.
[EINVAL] Op is equal to PROCLOCK and a process lock, a text lock, or a data lock already exists on the calling process.
[EINVAL] Op is equal to TXTLOCK and a text lock, or a process lock already exists on the calling process.
[EINVAL] Op is equal to DATLOCK and a data lock, or a process lock already exists on the calling process.
[EINVAL] Op is equal to UNLOCK and no type of lock exists on the calling process.
[EAGAIN] Not enough memory.

SEE ALSO
exec(2), exit(2), fork(2).

DIAGNOSTICS
Upon successful completion, a value of 0 is returned to the calling process. Otherwise, a value of -1 is returned and errno is set to indicate the error.
NAME
poll — STREAMS input/output multiplexing

SYNOPSIS
#include <stropts.h>
#include <poll.h>
int poll(fds, nfds, timeout)
struct pollfd fds[];
unsigned long nfds;
int timeout;

DESCRIPTION
poll provides users with a mechanism for multiplexing input/output over a set of
file descriptors that reference open streams [see intro(2)]. poll identifies those
streams on which a user can send or receive messages, or on which certain
events have occurred. A user can receive messages using read(2) or getmsg(2)
and can send messages using write(2) and putmsg(2). Certain ioctl(2) calls, such
as I_RECVFD and I_SENDFD [see streamio(7)], can also be used to receive and send
messages.

Fds specifies the file descriptors to be examined and the events of interest for
each file descriptor. It is a pointer to an array with one element for each open
file descriptor of interest. The array’s elements are pollfd structures which con­
tain the following members:

int fd;        /* file descriptor */
short events;  /* requested events */
short revents; /* returned events */

where fd specifies an open file descriptor and events and revents are bitmasks
constructed by or-ing any combination of the following event flags:

POLLIN A non-priority or file descriptor passing message (see I_RECVFD) is
present on the stream head read queue. This flag is set even if the
message is of zero length. In revents, this flag is mutually exclusive
with POLLPRI.

POLLPRI A priority message is present on the stream head read queue. This
flag is set even if the message is of zero length. In revents, this flag
is mutually exclusive with POLLIN.

POLLOUT The first downstream write queue in the stream is not full. Priority
control messages can be sent (see putmsg) at any time.

POLLErr An error message has arrived at the stream head. This flag is only
valid in the revents bitmask; it is not used in the events field.

POLLHUP A hangup has occurred on the stream. This event and POLLOUT
are mutually exclusive; a stream can never be writable if a hangup
has occurred. However, this event and POLLIN or POLLPRI are not
mutually exclusive. This flag is only valid in the revents bitmask; it
is not used in the events field.
POLL(2) POLL(2)

POLLNVAL  The specified *fd* value does not belong to an open *stream*. This flag is only valid in the *revents* field; it is not used in the *events* field.

For each element of the array pointed to by *fds*, *poll* examines the given file descriptor for the event(s) specified in *events*. The number of file descriptors to be examined is specified by *nfds*. If *nfds* exceeds NOFILES, the system limit of open files [see ulimit(2)], *poll* will fail.

If the value *fd* is less than zero, *events* is ignored and *revents* is set to 0 in that entry on return from *poll*.

The results of the *poll* query are stored in the *revents* field in the *pollfd* structure. Bits are set in the *revents* bitmask to indicate which of the requested events are true. If none are true, none of the specified bits is set in *revents* when the *poll* call returns. The event flags POLLHUP, POLLERR and POLLNVAL are always set in *revents* if the conditions they indicate are true; this occurs even though these flags were not present in *events*.

If none of the defined events have occurred on any selected file descriptor, *poll* waits at least *timeout* msec for an event to occur on any of the selected file descriptors. On a computer where millisecond timing accuracy is not available, *timeout* is rounded up to the nearest legal value available on that system. If the value *timeout* is 0, *poll* returns immediately. If the value of *timeout* is -1, *poll* blocks until a requested event occurs or until the call is interrupted. *poll* is not affected by the O_NDELAY flag.

*poll* fails if one or more of the following are true:

- **EAGAIN**  Allocation of internal data structures failed but request should be attempted again.
- **EFAULT**  Some argument points outside the allocated address space.
- **EINVAL**  The argument *nfds* is less than zero, or *nfds* is greater than NOFILES.

SEE ALSO

intro(2), read(2), getmsg(2), putmsg(2), write(2).


STREAMS Primer.

STREAMS Programmer's Guide.

DIAGNOSTICS

Upon successful completion, a non-negative value is returned. A positive value indicates the total number of file descriptors that has been selected (i.e., file descriptors for which the *revents* field is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, a value of -1 is returned and *errno* is set to indicate the error.

191
NAME
profil – execution time profile

SYNOPSIS
void profil (buff, bufsiz, offset, scale)
char *buff;
int bufsiz, offset, scale;

DESCRIPTION
Buff points to an area of core whose length (in bytes) is given by bufsiz. After this call, the user’s program counter (pc) is examined each clock tick. Then the value of offset is subtracted from it, and the remainder multiplied by scale. If the resulting number corresponds to an entry inside buff, that entry is incremented. An entry is defined as a series of bytes with length sizeof(short).

The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0177777 (octal) gives a 1-1 mapping of pc’s to entries in buff; 0777777 (octal) maps each pair of instruction entries together. 02(octal) maps all instructions onto the beginning of buff (producing a non-interrupting core clock).

Profiling is turned off by giving a scale of 0 or 1. It is rendered ineffective by giving a bufsiz of 0. Profiling is turned off when an exec is executed, but remains on in child and parent both after a fork. Profiling will be turned off if an update in buff would cause a memory fault.

SEE ALSO
prof(1), times(2), monitor(3C).

DIAGNOSTICS
Not defined.
NAME
ptrace — process trace

SYNOPSIS
int ptrace (request, pid, addr, data);
int request, pid, addr, data;

DESCRIPTION
ptrace provides a means by which a parent process may control the execution of a child process. Its primary use is for the implementation of breakpoint debugging [see sdb(1)]. The child process behaves normally until it encounters a signal [see signal(2) for the list], at which time it enters a stopped state and its parent is notified via wait(2). When the child is in the stopped state, its parent can examine and modify its "core image" using ptrace. Also, the parent can cause the child either to terminate or continue, with the possibility of ignoring the signal that caused it to stop.

The request argument determines the precise action to be taken by ptrace and is one of the following:

  0  This request must be issued by the child process if it is to be traced by its parent. It turns on the child's trace flag that stipulates that the child should be left in a stopped state upon receipt of a signal rather than the state specified by func [see signal(2)]. The pid, addr, and data arguments are ignored, and a return value is not defined for this request. Peculiar results will ensue if the parent does not expect to trace the child.

The remainder of the requests can only be used by the parent process. For each, pid is the process ID of the child. The child must be in a stopped state before these requests are made.

  1, 2  With these requests, the word at location addr in the address space of the child is returned to the parent process. If I and D space are separated, request 1 returns a word from I space, and request 2 returns a word from D space. If I and D space are not separated, either request 1 or request 2 may be used with equal results. The data argument is ignored. These two requests will fail if addr is not the start address of a word, in which case a value of -1 is returned to the parent process and the parent's errno is set to EIO.

  3  With this request, the word at location addr in the child's USER area in the system's address space (see <sys/user.h>) is returned to the parent process. The data argument is ignored. This request will fail if addr is not the start address of a word or is outside the USER area, in which case a value of -1 is returned to the parent process and the parent's errno is set to EIO.

  4, 5  With these requests, the value given by the data argument is written into the address space of the child at location addr. If I and D space are separated, request 4 writes a word into I space, and request 5 writes a word into D space. If I and D space are not separated, either request 4 or request 5 may be used with equal results. Upon successful completion, the value written into the
address space of the child is returned to the parent. These two requests will fail if \textit{addr} is not the start address of a word. Upon failure a value of \textit{-1} is returned to the parent process and the parent's \textit{errno} is set to \texttt{EIO}.

6 With this request, a few entries in the child's USER area can be written. \textit{Data} gives the value that is to be written and \textit{addr} is the location of the entry. The few entries that can be written are:

- the general registers
- the condition codes of the Processor Status Word.

7 This request causes the child to resume execution. If the \textit{data} argument is 0, all pending signals including the one that caused the child to stop are canceled before it resumes execution. If the \textit{data} argument is a valid signal number, the child resumes execution as if it had incurred that signal, and any other pending signals are canceled. The \textit{addr} argument must be equal to 1 for this request. Upon successful completion, the value of \textit{data} is returned to the parent. This request will fail if \textit{data} is not 0 or a valid signal number, in which case a value of \textit{-1} is returned to the parent process and the parent's \textit{errno} is set to \texttt{EIO}.

8 This request causes the child to terminate with the same consequences as \texttt{exit(2)}.

9 This request sets the trace bit in the Processor Status Word of the child and then executes the same steps as listed above for request 7. The trace bit causes an interrupt upon completion of one machine instruction. This effectively allows single stepping of the child.

To forestall possible fraud, \texttt{ptrace} inhibits the set-user-id facility on subsequent \texttt{exec(2)} calls. If a traced process calls \texttt{exec}, it will stop before executing the first instruction of the new image showing signal \texttt{SIGTRAP}.

General Errors
\texttt{ptrace} will in general fail if one or more of the following are true:

- [\texttt{EIO}] \textit{Request} is an illegal number.
- [\texttt{ESRCH}] \textit{Pid} identifies a child that does not exist or has not executed a \texttt{ptrace} with request 0.

SEE ALSO
\texttt{sdb(1), exec(2), signal(2), wait(2)}. 
NAME
putmsg — send a message on a stream

SYNOPSIS
#include <stropts.h>
int putmsg (fd, ctlptr, dataptr, flags)
int fd;
struct strbuf *ctlptr;
struct strbuf *dataptr;
int flags;

DESCRIPTION
putmsg creates a message [see intro(2)] from user specified buffer(s) and sends
the message to a STREAMS file. The message may contain either a data part, a
control part or both. The data and control parts to be sent are distinguished by
placement in separate buffers, as described below. The semantics of each part is
defined by the STREAMS module that receives the message.

fd specifies a file descriptor referencing an open stream. ctlptr and dataptr each
point to a strbuf structure which contains the following members:

    int maxlen;        /* not used */
    int len;           /* length of data */
    char *buf;         /* ptr to buffer */

ctlptr points to the structure describing the control part, if any, to be included in
the message. The buf field in the strbuf structure points to the buffer where the
control information resides, and the len field indicates the number of bytes to be
sent. The maxlen field is not used in putmsg [see getmsg(2)]. In a similar
manner, dataptr specifies the data, if any, to be included in the message. flags
may be set to the values 0 or RS_HIPRI and is used as described below.

To send the data part of a message, dataptr must be non-NULL and the len field
of dataptr must have a value of 0 or greater. To send the control part of a mes­
sage, the corresponding values must be set for ctlptr. No data (control) part will
be sent if either dataptr (ctlptr) is NULL or the len field of dataptr (ctlptr) is set to
-1.

If a control part is specified, and flags is set to RS_HIPRI, a priority message is
sent. If flags is set to 0, a non-priority message is sent. If no control part is
specified, and flags is set to RS_HIPRI, putmsg fails and sets errno to EINVAL. If
no control part and no data part are specified, and flags is set to 0, no message is
sent, and 0 is returned.

For non-priority messages, putmsg will block if the stream write queue is full due
to internal flow control conditions. For priority messages, putmsg does not block
on this condition. For non-priority messages, putmsg does not block when the
write queue is full and O_NDELAY is set. Instead, it fails and sets errno to
EAGAIN.

putmsg 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.
putmsg fails if one or more of the following are true:

[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.

[EAGAIN] Buffers could not be allocated for the message that was to be created.

[EBADF] `fd` is not a valid file descriptor open for writing.

[EFAULT] `ctlptr` or `dataptr` points outside the allocated address space.

[EINTR] A signal was caught during the `putmsg` system call.

[EINVAL] An undefined value was specified in `flags`, or `flags` is set to RS_HIPRI and no control part was supplied.

[EINVAL] The `stream` referenced by `fd` is linked below a multiplexor.

[ENOISTR] A `stream` is not associated with `fd`.

[ENXIO] A hangup condition was generated downstream for the specified `stream`.

[ERANGE] The size of the data part of the message does not fall within the range specified by the maximum and minimum packet sizes of the topmost `stream` module. This value is also returned if the control part of the message is larger than the maximum configured size of the control part of a message, or if the data part of a message is larger than the maximum configured size of the data part of a message.

A `putmsg` also fails if a STREAMS error message had been processed by the `stream` head before the call to `putmsg`. The error returned is the value contained in the STREAMS error message.

SEE ALSO

intro(2), read(2), getmsg(2), poll(2), write(2).
STREAMS Primer.
STREAMS Programmer's Guide.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.
NAME
read — read from file

SYNOPSIS

```c
int read (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

DESCRIPTION

`fildes` is a file descriptor obtained from a `creat(2), open(2), dup(2),fcntl(2),` or `pipe(2)` system call.

`read` attempts to read `nbyte` bytes from the file associated with `fildes` into the buffer pointed to by `buf`.

On devices capable of seeking, the `read` starts at a position in the file given by the file pointer associated with `fildes`. Upon return from `read`, the file pointer is incremented by the number of bytes actually read.

Devices that are incapable of seeking always read from the current position. The value of a file pointer associated with such a file is undefined.

Upon successful completion, `read` returns the number of bytes actually read and placed in the buffer; this number may be less than `nbyte` if the file is associated with a communication line [see `ioctl(2)` and `termio(7)`], or if the number of bytes left in the file is less than `nbyte` bytes. A value of 0 is returned when an end-of-file has been reached.

A `read` from a STREAMS [see `intro(2)`] file can operate in three different modes: "byte-stream" mode, "message-nondiscard" mode, and "message-discard" mode.

The default is byte-stream mode. This can be changed using the `I_SRDOPT ioctl` request [see `streamio(7)`], and can be tested with the `I_GRDOPT ioctl`. In byte-stream mode, `read` will retrieve data from the `stream` until it has retrieved `nbyte` bytes, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, `read` retrieves data until it has read `nbyte` bytes, or until it reaches a message boundary. If the `read` does not retrieve all the data in a message, the remaining data are replaced on the `stream`, and can be retrieved by the next `read` or `getmsg(2)` call. Message-discard mode also retrieves data until it has retrieved `nbyte` bytes, or it reaches a message boundary. However, unread data remaining in a message after the `read` returns are discarded, and are not available for a subsequent `read` or `getmsg`.

When attempting to read from a regular file with mandatory file/record locking set [see `chmod(2)`], and there is a blocking (i.e. owned by another process) write lock on the segment of the file to be read:

- If `O_NDELAY` is set, the read will return a `-1` and set `errno` to `EAGAIN`.
- If `O_NDELAY` is clear, the read will sleep until the blocking record lock is removed.

When attempting to read from an empty pipe (or FIFO):
If O_NDELAY is set, the read will return a 0.
If O_NDELAY is clear, the read will block until data is written to the file
or the file is no longer open for writing.

When attempting to read a file associated with a tty that has no data currently
available:
If O_NDELAY is set, the read will return a 0.
If O_NDELAY is clear, the read will block until data becomes available.

When attempting to read a file associated with a stream that has no data
currently available:
If O_NDELAY is set, the read will return a -1 and set errno to EAGAIN.
If O_NDELAY is clear, the read will block until data becomes available.

When reading from a STREAMS file, handling of zero-byte messages is deter­
mined by the current read mode setting. In byte-stream mode, read accepts data
until it has read nbytes bytes, or until there is no more data to read, or until a
zero-byte message block is encountered. read then returns the number of bytes
read, and places the zero-byte message back on the stream to be retrieved by the
next read or getmsg. In the two other modes, a zero-byte message returns a
value of 0 and the message is removed from the stream. When a zero-byte mes­
sage is read as the first message on a stream, a value of 0 is returned regardless
of the read mode.

A read from a STREAMS file can only process data messages. It cannot process
any type of protocol message and will fail if a protocol message is encountered
at the stream head.

read will fail if one or more of the following are true:

[EAGAIN] Mandatory file/record locking was set, O_NDELAY was set,
and there was a blocking record lock.

[EAGAIN] Total amount of system memory available when reading via
raw IO is temporarily insufficient.

[EAGAIN] No message waiting to be read on a stream and O_NDELAY flag
set.

[EBADF] Fildes is not a valid file descriptor open for reading.

[EBADMSG] Message waiting to be read on a stream is not a data message.

[EDEADLK] The read was going to go to sleep and cause a deadlock situa­
tion to occur.

[EFAULT] Buf points outside the allocated address space.

[EINTR] A signal was caught during the read system call.

[EINVAL] Attempted to read from a stream linked to a multiplexor.

[ENOLCK] The system record lock table was full, so the read could not go
to sleep until the blocking record lock was removed.

[ENOLINK] Fildes is on a remote machine and the link to that machine is no
longer active.
A read from a STREAMS file will also fail if an error message is received at the stream head. In this case, errno is set to the value returned in the error message. If a hangup occurs on the stream being read, read will continue to operate normally until the stream head read queue is empty. Thereafter, it will return 0.

SEE ALSO

DIAGNOSTICS
Upon successful completion a non-negative integer is returned indicating the number of bytes actually read. Otherwise, a -1 is returned and errno is set to indicate the error.
NAME
   rmdir — remove a directory

SYNOPSIS
   int rmdir (path)
   char *path;

DESCRIPTION
   rmdir removes the directory named by the path name pointed to by path. The
directory must not have any entries other than "." and ". ..".
The named directory is removed unless one or more of the following are true:

   [EINVAL]  The current directory may not be removed.
   [EINVAL]  The "." entry of a directory may not be removed.
   [EEXIST]  The directory contains entries other than those for "." and ". ..".
   [ENOTDIR] A component of the path prefix is not a directory.
   [ENOENT]  The named directory does not exist.
   [EACCES]  Search permission is denied for a component of the path prefix.
   [EACCES]  Write permission is denied on the directory containing the
directory to be removed.
   [EBUSY]   The directory to be removed is the mount point for a mounted
file system.
   [EROFS]   The directory entry to be removed is part of a read-only file
system.
   [EFAULT]  Path points outside the process's allocated address space.
   [EIO]     An I/O error occurred while accessing the file system.
   [ENOLINK] Path points to a remote machine, and the link to that machine
is no longer active.
   [EMULTIHOP] Components of path require hopping to multiple remote
machines.

DIAGNOSTICS
   Upon successful completion, a value of 0 is returned. Otherwise, a value of -1
is returned and errno is set to indicate the error.

SEE ALSO
   mkdir(2).
   rmdir(1), rm(1), and mkdir(1) in the User's Reference Manual.
NAME
semctl — semaphore control operations

SYNOPSIS
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl (semid, semnum, cmd, arg)
int semid, cmd;
int semnum;
union semun {
    int val;
    struct semid_ds *buf;
    ushort *array;
} arg;

DESCRIPTION
semctl provides a variety of semaphore control operations as specified by cmd.
The following cmds are executed with respect to the semaphore specified by semid and semnum:

GETVAL Return the value of semval [see intro(2)]. {READ}
SETVAL Set the value of semval to arg.val. {ALTER} When this cmd is successfully executed, the semadj value corresponding to the specified semaphore in all processes is cleared.
GETPID Return the value of sempid. {READ}
GETNCNT Return the value of semncnt. {READ}
GETZCNT Return the value of semzcnt. {READ}

The following cmds return and set, respectively, every semval in the set of semaphores.

GETALL Place semvals into array pointed to by arg.array. {READ}
SETALL Set semvals according to the array pointed to by arg.array. {ALTER} When this cmd is successfully executed the semadj values corresponding to each specified semaphore in all processes are cleared.

The following cmds are also available:

IPC_STAT Place the current value of each member of the data structure associated with semid into the structure pointed to by arg.buf. The contents of this structure are defined in intro(2). {READ}

IPC_SET Set the value of the following members of the data structure associated with semid to the corresponding value found in the structure pointed to by arg.buf:
sem_perm.uid
sem_perm.gid
sem_perm.mode /* only low 9 bits */
This cmd can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of \texttt{sem_perm.cuid} or \texttt{sem_perm.uid} in the data structure associated with \texttt{semid}.

**IPC_RMID** Remove the semaphore identifier specified by \texttt{semid} from the system and destroy the set of semaphores and data structure associated with it. This cmd can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of \texttt{sem_perm.cuid} or \texttt{sem_perm.uid} in the data structure associated with \texttt{semid}.

\texttt{semctl} fails if one or more of the following are true:

- [EINVAL] \texttt{Semid} is not a valid semaphore identifier.
- [EINVAL] \texttt{Semnum} is less than zero or greater than \texttt{sem_nsems}.
- [EINVAL] \texttt{Cmd} is not a valid command.
- [EACCES] Operation permission is denied to the calling process [see \texttt{intra}(2)].
- [ERANGE] \texttt{Cmd} is \texttt{SETVAL} or \texttt{SETALL} and the value to which \texttt{semval} is to be set is greater than the system imposed maximum.
- [EPERM] \texttt{Cmd} is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of super-user, or to the value of \texttt{sem_perm.cuid} or \texttt{sem_perm.uid} in the data structure associated with \texttt{semid}.
- [EFAULT] \texttt{Arg.buf} points to an illegal address.

**SEE ALSO**
\texttt{intro}(2), \texttt{semget}(2), \texttt{semop}(2).

**DIAGNOSTICS**
Upon successful completion, the value returned depends on \texttt{cmd} as follows:

\begin{itemize}
  \item \textbf{GETVAL} \texttt{The value of semval.}
  \item \textbf{GETPID} \texttt{The value of sempid.}
  \item \textbf{GETNCNT} \texttt{The value of semncnt.}
  \item \textbf{GETZCNT} \texttt{The value of semzcnt.}
  \item All others \texttt{A value of 0.}
\end{itemize}

Otherwise, a value of \texttt{-1} is returned and \texttt{errno} is set to indicate the error.
SEMGET(2)

NAME
semget — get set of semaphores

SYNOPSIS
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget (key, nsems, semflg)
key_t key;
int nsems, semflg;

DESCRIPTION
semget returns the semaphore identifier associated with key.

A semaphore identifier and associated data structure and set containing nsems
semaphores [see intro(2)] are created for key if one of the following is true:

Key is equal to IPC_PRIVATE.
Key does not already have a semaphore identifier associated with it, and
(semflg & IPC_CREAT) is "true".

Upon creation, the data structure associated with the new semaphore identifier is
initialized as follows:

Sem_perm.cuid, sem_perm.uid, sem_perm.cgid, and sem_perm.gid are
set equal to the effective user ID and effective group ID, respectively, of
the calling process.
The low-order 9 bits of sem_perm.mode are set equal to the low-order 9
bits of semflg.

Sem_nsems is set equal to the value of nsems.

Sem_otime is set equal to 0 and sem_ctime is set equal to the current
time.

semget fails if one or more of the following are true:

[EINVAL] Nsems is either less than or equal to zero or greater than the
system-imposed limit.

[EACCES] A semaphore identifier exists for key, but operation permission
[see intro(2)] as specified by the low-order 9 bits of semflg would
not be granted.

[EINVAL] A semaphore identifier exists for key, but the number of sema-
phores in the set associated with it is less than nsems, and nsems
is not equal to zero.

[ENOENT] A semaphore identifier does not exist for key and (semflg &
IPC_CREAT) is "false".

[ENOSPC] A semaphore identifier is to be created but the system-imposed
limit on the maximum number of allowed semaphore identifiers
system wide would be exceeded.
A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores system wide would be exceeded.

A semaphore identifier exists for key but ((semflg & IPC_CREAT) and (semflg & IPC_EXCL)) is “true”.

SEE ALSO
intro(2), semctl(2), semop(2).

DIAGNOSTICS
Upon successful completion, a non-negative integer, namely a semaphore identifier, is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.
SEMOP(2)

NAME
semop — semaphore operations

SYNOPSIS
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop (semid, sops, nsops)
int semid;
struct sembuf **sops;
unsigned nsops;

DESCRIPTION
semop is used to automatically perform an array of semaphore operations on the
set of semaphores associated with the semaphore identifier specified by semid.
Sops is a pointer to the array of semaphore-operation structures. Nsops is the
number of such structures in the array. The contents of each structure includes
the following members:

  short sem_num;  /* semaphore number */
  short sem_op;   /* semaphore operation */
  short sem_flg;  /* operation flags */

Each semaphore operation specified by sem_op is performed on the
 corresponding semaphore specified by semid and sem_num.

Sem_op specifies one of three semaphore operations as follows:

If sem_op is a negative integer, one of the following will occur: [ALTER]

If semval [see intro(2)] is greater than or equal to the absolute
value of sem_op, the absolute value of sem_op is subtracted from
semval. Also, if (sem_flg & SEM_UNDO) is "true", the absolute
value of sem_op is added to the calling process's semadj value
[see exit(2)] for the specified semaphore.

If semval is less than the absolute value of sem_op and (sem_flg
& IPC_NOWAIT) is "true", semop will return immediately.

If semval is less than the absolute value of sem_op and (sem_flg
& IPC_NOWAIT) is "false", semop will increment the semcnt
associated with the specified semaphore and suspend execution
of the calling process until one of the following conditions
occur.

Semval becomes greater than or equal to the absolute value of
sem_op. When this occurs, the value of semcnt associated
with the specified semaphore is decremented; the absolute
value of sem_op is subtracted from semval and, if (sem_flg &
SEM_UNDO) is "true", the absolute value of sem_op is added to
the calling process's semadj value for the specified semaphore.

The semid for which the calling process is awaiting action is
removed from the system [see semctl(2)]. When this occurs,
errno is set equal to EIDRM, and a value of -1 is returned.
The calling process receives a signal that is to be caught. When this occurs, the value of semncnt associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in signal(2).

If `sem_op` is a positive integer, the value of `sem_op` is added to `semval` and, if `(sem_flg & SEM_UNDO)` is "true", the value of `sem_op` is subtracted from the calling process's semadj value for the specified semaphore. (ALTER)

If `sem_op` is zero, one of the following will occur: {READ}

If `semval` is zero, `semop` will return immediately.

If `semval` is not equal to zero and `(sem_flg & IPC_NOWAIT)` is "true", `semop` will return immediately.

If `semval` is not equal to zero and `(sem_flg & IPC_NOWAIT)` is "false", `semop` will increment the semzcnt associated with the specified semaphore and suspend execution of the calling process until one of the following occurs:

Semval becomes zero, at which time the value of semzcnt associated with the specified semaphore is decremented.

The semid for which the calling process is awaiting action is removed from the system. When this occurs, `errno` is set equal to EIDRM, and a value of −1 is returned.

The calling process receives a signal that is to be caught. When this occurs, the value of semzcnt associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in signal(2).

`semop` will fail if one or more of the following are true for any of the semaphore operations specified by `sops`:

- **[EINVAL]** `Semid` is not a valid semaphore identifier.
- **[EFAULT]** `Sem_num` is less than zero or greater than or equal to the number of semaphores in the set associated with `semid`.
- **[E2BIG]** `Nsops` is greater than the system-imposed maximum.
- **[EACCES]** Operation permission is denied to the calling process [see intro(2)]
- **[EAGAIN]** The operation would result in suspension of the calling process but `(sem_flg & IPC_NOWAIT)` is "true".
- **[ENOSPC]** The limit on the number of individual processes requesting an `SEM_UNDO` would be exceeded.
- **[EINVAL]** The number of individual semaphores for which the calling process requests a `SEM_UNDO` would exceed the limit.
- **[ERANGE]** An operation would cause a semval to overflow the system-imposed limit.
An operation would cause a semadj value to overflow the system-imposed limit.

Sops points to an illegal address.

Upon successful completion, the value of sempid for each semaphore specified in the array pointed to by sops is set equal to the process ID of the calling process.

SEE ALSO
exec(2), exit(2), fork(2), intro(2), semctl(2), semget(2).

DIAGNOSTICS
If semop returns due to the receipt of a signal, a value of −1 is returned to the calling process and errno is set to EINTR. If it returns due to the removal of a semid from the system, a value of −1 is returned and errno is set to EIDRM.

Upon successful completion, a value of zero is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.
NAME
   setpgp – set process group ID
SYNOPSIS
   int setpgp ()
DESCRIPTION
   setpgp sets the process group ID of the calling process to the process ID of the
calling process and returns the new process group ID.
SEE ALSO
   exec(2), fork(2), getpid(2), intro(2), kill(2), signal(2).
DIAGNOSTICS
   setpgp returns the value of the new process group ID.
NAME
setuid, setgid — set user and group IDs

SYNOPSIS

    int setuid (uid)
    int uid;

    int setgid (gid)
    int gid;

DESCRIPTION

setuid (setgid) is used to set the real user (group) ID and effective user (group) ID
of the calling process.

If the effective user ID of the calling process is super-user, the real user (group)
ID and effective user (group) ID are set to uid (gid).

If the effective user ID of the calling process is not super-user, but its real user
(group) ID is equal to uid (gid), the effective user (group) ID is set to uid (gid).

If the effective user ID of the calling process is not super-user, but the saved set-
user (group) ID from exec(2) is equal to uid (gid), the effective user (group) ID is
set to uid (gid).

setuid (setgid) will fail if the real user (group) ID of the calling process is not
equal to uid (gid) and its effective user ID is not super-user. [EPERM]

The uid is out of range. [EINVAL]

SEE ALSO

getuid(2), intro(2).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1
is returned and errno is set to indicate the error.
NAME
shmct1 - shared memory control operations

SYNOPSIS
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmct1 (shmid, cmd, buf)
int shmid, cmd;
struct shmid_ds *buf;

DESCRIPTION
shmct1 provides a variety of shared memory control operations as specified by cmd. The following commands are available:

IPC_STAT Place the current value of each member of the data structure associated with shmid into the structure pointed to by buf. The contents of this structure are defined in intro(2). [READ]

IPC_SET Set the value of the following members of the data structure associated with shmid to the corresponding value found in the structure pointed to by buf:

shm_perm.uid
shm_perm.gid
shm_perm.mode /* only low 9 bits */

This cmd can only be executed by a process that has an effective user ID equal to that of super user, or to the value of shm_perm.cuid or shm_perm.uid in the data structure associated with shmid.

IPC_RMID Remove the shared memory identifier specified by shmid from the system and destroy the shared memory segment and data structure associated with it. This cmd can only be executed by a process that has an effective user ID equal to that of super user, or to the value of shm_perm.cuid or shm_perm.uid in the data structure associated with shmid.

SHM_LOCK Lock the shared memory segment specified by shmid in memory. This cmd can only be executed by a process that has an effective user ID equal to super user.

SHM_UNLOCK Unlock the shared memory segment specified by shmid. This cmd can only be executed by a process that has an effective user ID equal to super user.

shmct1 will fail if one or more of the following are true:

[EINVAL] Shmid is not a valid shared memory identifier.
[EINVAL] Cmd is not a valid command.
[EACCES] Cmd is equal to IPC_STAT and {READ} operation permission is denied to the calling process [see intro(2)].
SHMCTL(2) SHMCTL(2)

[EPERM] Cmd is equal to IPC_RMID or IPC_SET and the effective user ID of
the calling process is not equal to that of super user, or to the value
of shm_perm.cuid or shm_perm.uid in the data structure associ­
ated with shmid.

[EPERM] Cmd is equal to SHM_LOCK or SHM_UNLOCK and the effective user
ID of the calling process is not equal to that of super user.

[EFAULT] Buf points to an illegal address.

[ENOMEM] Cmd is equal to SHM_LOCK and there is not enough memory.

SEE ALSO
shmget(2), shmop(2).

DIAGNOSTICS
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is
returned and errno is set to indicate the error.

NOTES
The user must explicitly remove shared memory segments after the last reference
to them has been removed.
SHMGET(2)

NAME
shmget — get shared memory segment identifier

SYNOPSIS
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget (key, size, shmflg)
key_t key;
int size, shmflg;

DESCRIPTION
shmget returns the shared memory identifier associated with key.

A shared memory identifier and associated data structure and shared memory
segment of at least size bytes [see intro(2)] are created for key if one of the fol-
lowing are true:

Key is equal to IPC_PRIVATE.

Key does not already have a shared memory identifier associated with it,
and (shmflg & IPC_CREAT) is "true".

Upon creation, the data structure associated with the new shared memory
identifier is initialized as follows:

Shm_perm.cuid, shm_perm.uid, shm_perm.cgid, and shm_perm.gid
are set equal to the effective user ID and effective group ID, respectively,
of the calling process.

The low-order 9 bits of shm_perm.mode are set equal to the low-order
9 bits of shmflg. Shm_segsz is set equal to the value of size.

Shm_lpid, shm_nattch, shm_atime, and shm_dtime are set equal to 0.

Shm_ctime is set equal to the current time.

shmget will fail if one or more of the following are true:

[EINVAL] Size is less than the system-imposed minimum or greater than
the system-imposed maximum.

[EACCES] A shared memory identifier exists for key but operation permis-
sion [see intro(2)] as specified by the low-order 9 bits of shmflg
would not be granted.

[EINVAL] A shared memory identifier exists for key but the size of the
segment associated with it is less than size and size is not equal
to zero.

[ENOENT] A shared memory identifier does not exist for key and (shmflg &
IPC_CREAT) is "false".

[ENOSPC] A shared memory identifier is to be created but the system-
imposed limit on the maximum number of allowed shared
memory identifiers system wide would be exceeded.
A shared memory identifier and associated shared memory segment are to be created but the amount of available memory is not sufficient to fill the request.

A shared memory identifier exists for key but ((shmflg & IPC_CREAT) and (shmflg & IPC_EXCL)) is "true".

SEE ALSO
intro(2), shmctl(2), shmop(2).

DIAGNOSTICS
Upon successful completion, a non-negative integer, namely a shared memory identifier is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

NOTES
The user must explicitly remove shared memory segments after the last reference to them has been removed.
NAME
shmop — shared memory operations

SYNOPSIS
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

char *shmat (shmid, shmaddr, shmflg)
int shmid;
char *shmaddr;
int shmflg;

int shmdt (shmaddr)
char *shmaddr;

DESCRIPTION

Shmat attaches the shared memory segment associated with the shared memory identifier specified by shmid to the data segment of the calling process. The segment is attached at the address specified by one of the following criteria:

If shmaddr is equal to zero, the segment is attached at the first available address as selected by the system.

If shmaddr is not equal to zero and (shmflg & SHM_RND) is "true", the segment is attached at the address given by (shmaddr - (shmaddr modulus SHMLBA)).

If shmaddr is not equal to zero and (shmflg & SHM_RND) is "false", the segment is attached at the address given by shmaddr.

Shmdt detaches from the calling process’s data segment the shared memory segment located at the address specified by shmaddr.

The segment is attached for reading if (shmflg & SHM_RDONLY) is "true" [READ], otherwise it is attached for reading and writing [READ/WRITE].

Shmat will fail and not attach the shared memory segment if one or more of the following are true:

[EINVAL] Shmid is not a valid shared memory identifier.

[EACCES] Operation permission is denied to the calling process [see intro(2)].

[ENOMEM] The available data space is not large enough to accommodate the shared memory segment.

[EINVAL] Shmaddr is not equal to zero, and the value of (shmaddr - (shmaddr modulus SHMLBA)) is an illegal address.

[EINVAL] Shmaddr is not equal to zero, (shmflg & SHM_RND) is "false", and the value of shmaddr is an illegal address.

[EMFILE] The number of shared memory segments attached to the calling process would exceed the system-imposed limit.

[EINVAL] Shmdt will fail and not detach the shared memory segment if shmaddr is not the data segment start address of a shared
memory segment.

SEE ALSO
exec(2), exit(2), fork(2), intro(2), shmct1(2), shmget(2).

DIAGNOSTICS
Upon successful completion, the return value is as follows:

Shmat returns the data segment start address of the attached shared memory segment.

Shmdt returns a value of 0.

Otherwise, a value of -1 is returned and errno is set to indicate the error.

NOTES
The user must explicitly remove shared memory segments after the last reference to them has been removed.
NAME
signal — specify what to do upon receipt of a signal

SYNOPSIS
#include <signal.h>

void (*signal (sig, func))();
int sig;
void (*func)();

DESCRIPTION
signal allows the calling process to choose one of three ways in which it is possible to handle the receipt of a specific signal. Sig specifies the signal and func specifies the choice.

Sig can be assigned any one of the following except SIGKILL:

- SIGHUP 01 hangup
- SIGINT 02 interrupt
- SIGQUIT 03[1] quit
- SIGILL 04[1] illegal instruction (not reset when caught)
- SIGTRAP 05[1] trace trap (not reset when caught)
- SIGIOT 06[1] IOT instruction
- SIGEMT 07[1] EMT instruction
- SIGFPE 08[1] floating point exception
- SIGKILL 09 kill (cannot be caught or ignored)
- SIGBUS 10[1] bus error
- SIGSEGV 11[1] segmentation violation
- SIGSYS 12[1] bad argument to system call
- SIGPIPE 13 write on a pipe with no one to read it
- SIGALRM 14 alarm clock
- SIGTERM 15 software termination signal
- SIGUSR1 16 user-defined signal 1
- SIGUSR2 17 user-defined signal 2
- SIGCLD 18[2] death of a child
- SIGPWR 19[2] power fail
- SIGPOLL 22[3] selectable event pending

Func is assigned one of three values: SIG_DFL, SIG_IGN, or a function address. SIG_DFL and SIG_IGN, are defined in the include file signal.h. Each is a macro that expands to a constant expression of type pointer to function returning void, and has a unique value that matches no declarable function.

The actions prescribed by the values of func are as follows:

SIG_DFL — terminate process upon receipt of a signal
Upon receipt of the signal sig, the receiving process is to be terminated with all of the consequences outlined in exit(2). See NOTE [1] below.

SIG_IGN — ignore signal
The signal sig is to be ignored.

Note: the signal SIGKILL cannot be ignored.
function address — catch signal

Upon receipt of the signal sig, the receiving process is to execute the signal-catching function pointed to by func. The signal number sig will be passed as the only argument to the signal-catching function. Additional arguments are passed to the signal-catching function for hardware-generated signals. Before entering the signal-catching function, the value of func for the caught signal will be set to SIG_DFL unless the signal is SIGILL, SIGTRAP, or SIGPWR.

Upon return from the signal-catching function, the receiving process will resume execution at the point it was interrupted.

When a signal that is to be caught occurs during a read(2), a write(2), an open(2), or an ioctl(2) system call on a slow device (like a terminal; but not a file), during a pause(2) system call, or during a wait(2) system call that does not return immediately due to the existence of a previously stopped or zombie process, the signal catching function will be executed and then the interrupted system call may return a −1 to the calling process with errno set to EINTR.

signal will not catch an invalid function argument, func, and results are undefined when an attempt is made to execute the function at the bad address.

Note: The signal SIGKILL cannot be caught.

A call to signal cancels a pending signal sig except for a pending SIGKILL signal. signal will fail if sig is an illegal signal number, including SIGKILL. [EINVAL]

NOTES

1. If SIG_DFL is assigned for these signals, in addition to the process being terminated, a “core image” will be constructed in the current working directory of the process, if the following conditions are met:

The effective user ID and the real user ID of the receiving process are equal.

An ordinary file named core exists and is writable or can be created. If the file must be created, it will have the following properties:

- a mode of 0666 modified by the file creation mask [see umask(2)]
- a file owner ID that is the same as the effective user ID of the receiving process.
- a file group ID that is the same as the effective group ID of the receiving process

2. For the signals SIGCLD and SIGPWR, func is assigned one of three values: SIG_DFL, SIG_IGN, or a function address. The actions prescribed by these values are:

SIG_DFL - ignore signal
The signal is to be ignored.
SIG_Ign - ignore signal
The signal is to be ignored. Also, if sig is SIGCLD, the calling
process's child processes will not create zombie processes when
they terminate [see exit(2)].

function address - catch signal
If the signal is SIGPWR, the action to be taken is the same as that
described above for func equal to function address. The same is
ture if the signal is SIGCLD with one exception: while the process
is executing the signal-catching function, any received SIGCLD
signals will be ignored. (This is the default action.)

In addition, SIGCLD affects the wait, and exit system calls as follows:

wait  If the func value of SIGCLD is set to SIG_IGN and a wait is exe-
cuted, the wait will block until all of the calling process's child
processes terminate; it will then return a value of -1 with errno
set to ECHILD.

exit  If in the exiting process's parent process the func value of
SIGCLD is set to SIG_IGN, the exiting process will not create a
zombie process.

When processing a pipeline, the shell makes the last process in the pipe-
line the parent of the proceeding processes. A process that may be piped
into in this manner (and thus become the parent of other processes)
should take care not to set SIGCLD to be caught.

SIGPOLL is issued when a file descriptor corresponding to a STREAMS [see
intro(2)] file has a "selectable" event pending. A process must specifically
request that this signal be sent using the I_SETSIG ioctl call. Otherwise, the
process will never receive SIGPOLL.

SEE ALSO
intro(2), kill(2), pause(2), ptrace(2), wait(2), setjmp(3C), sigset(2).

DIAGNOSTICS
Upon successful completion, signal returns the previous value of func for the
specified signal sig. Otherwise, a value of SIG_ERR is returned and errno is set
to indicate the error. SIG_ERR is defined in the include file signal.h.
NAME
sigset, sighold, sigrelse, sigignore, sigpause — signal management

SYNOPSIS
#include <signal.h>
void (*sigset (sig, func))()
int sig;
void (*func)();
int sighold (sig)
int sig;
int sigrelse (sig)
int sig;
int sigignore (sig)
int sig;
int sigpause (sig)
int sig;

DESCRIPTION
These functions provide signal management for application processes. sigset
specifies the system signal action to be taken upon receipt of signal sig. This
action is either calling a process signal-catching handler func or performing a
system-defined action.

Sig can be assigned any one of the following values except SIGKILL. Machine or
implementation dependent signals are not included (see NOTES below). Each
value of sig is a macro, defined in <signal.h>, that expands to an integer con­
stant expression.

SIGHUP    hangup
SIGINT    interrupt
SIGQUIT*  quit
SIGILL*   illegal instruction (not held when caught)
SIGTRAP*  trace trap (not held when caught)
SIGABRT*  abort
SIGFPE*   floating point exception
SIGKILL   kill (can not be caught or ignored)
SIGSYS*   bad argument to system call
SIGPIPE   write on a pipe with no one to read it
SIGALRM   alarm clock
SIGTERM   software termination signal
SIGUSR1   user-defined signal 1
SIGUSR2   user-defined signal 2
SIGCHLD   death of a child (see WARNING below)
SIGPWR    power fail (see WARNING below)
SIGPOLL   selectable event pending (see NOTES below)

See below under SIG_DFL regarding asterisks (*) in the above list.
The following values for the system-defined actions of *func* are also defined in `<signal.h>`. Each is a macro that expands to a constant expression of type pointer to function returning *void* and has a unique value that matches no declarable function.

**SIG_DFL** — default system action
Upon receipt of the signal *sig*, the receiving process is to be terminated with all of the consequences outlined in `exit(2)`. In addition a "core image" will be made in the current working directory of the receiving process if *sig* is one for which an asterisk appears in the above list *and* the following conditions are met:

The effective user ID and the real user ID of the receiving process are equal.

An ordinary file named *core* exists and is writable or can be created. If the file must be created, it will have the following properties:

- a mode of 0666 modified by the file creation mask [see `umask(2)`]
- a file owner ID that is the same as the effective user ID of the receiving process.
- a file group ID that is the same as the effective group ID of the receiving process.

**SIG_IGN** — ignore signal
Any pending signal *sig* is discarded and the system signal action is set to ignore future occurrences of this signal type.

**SIG_HOLD** — hold signal
The signal *sig* is to be held upon receipt. Any pending signal of this type remains held. Only one signal of each type is held.

Otherwise, *func* must be a pointer to a function, the signal-catch handler, that is to be called when signal *sig* occurs. In this case, *sigset* specifies that the process will call this function upon receipt of signal *sig*. Any pending signal of this type is released. This handler address is retained across calls to the other signal management functions listed here.

When a signal occurs, the signal number *sig* will be passed as the only argument to the signal-catch handler. Before calling the signal-catch handler, the system signal action will be set to SIG_HOLD. During normal return from the signal-catch handler, the system signal action is restored to *func* and any held signal of this type released. If a non-local goto (`longjmp`) is taken, then *sigrelse* must be called to restore the system signal action and release any held signal of this type.

In general, upon return from the signal-catch handler, the receiving process will resume execution at the point it was interrupted. However, when a signal is caught during a `read(2)`, a `write(2)`, an `open(2)`, or an `ioctl(2)` system call during a `sigpause` system call, or during a `wait(2)` system call that does not return immediately due to the existence of a previously stopped or zombie process, the signal-
catching handler will be executed and then the interrupted system call may return a −1 to the calling process with errno set to EINTR.

**Sighold** and **sigrelse** are used to establish critical regions of code. **Sighold** is analogous to raising the priority level and deferring or holding a signal until the priority is lowered by **sigrelse**. **Sigrelse** restores the system signal action to that specified previously by **sigset**.

**Sigignore** sets the action for signal **sig** to SIG_IGN (see above).

**Sigpause** suspends the calling process until it receives a signal, the same as **pause(2)**. However, if the signal **sig** had been received and held, it is released and the system signal action taken. This system call is useful for testing variables that are changed on the occurrence of a signal. The correct usage is to use **sighold** to block the signal first, then test the variables. If they have not changed, then call **sigpause** to wait for the signal. **sigset** will fail if one or more of the following are true:

- **[EINVAL]** Sig is an illegal signal number (including SIGKILL) or the default handling of **sig** cannot be changed.
- **[EINTR]** A signal was caught during the system call **sigpause**.

**DIAGNOSTICS**

Upon successful completion, **sigset** returns the previous value of the system signal action for the specified signal **sig**. Otherwise, a value of SIG_ERR is returned and errno is set to indicate the error. SIG_ERR is defined in `<signal.h>`.

For the other functions, upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

**SEE ALSO**

kill(2), pause(2), signal(2), wait(2), setjmp(3C).

**WARNING**

Two signals that behave differently than the signals described above exist in this release of the system:

- **SIGCLD** - death of a child (reset when caught)
- **SIGPWR** - power fail (not reset when caught)

For these signals, **func** is assigned one of three values: SIG_DFL, SIG_IGN, or a **function address**. The actions prescribed by these values are as follows:

- **SIG_DFL** - ignore signal
  The signal is to be ignored.

- **SIG_IGN** - ignore signal
  The signal is to be ignored. Also, if **sig** is SIGCLD, the calling process's child processes will not create zombie processes when they terminate [see exit(2)].

- **function address** - catch signal
  If the signal is SIGPWR, the action to be taken is the same as that described above for **func** equal to **function address**. The same is true
if the signal is SIGCLD with one exception: while the process is executing the signal-catching function, any received SIGCLD signals will be ignored. (This is the default action.)

The SIGCLD affects two other system calls [wait(2), and exit(2)] in the following ways:

**wait**  If the `func` value of SIGCLD is set to SIG_IGN and a `wait` is executed, the `wait` will block until all of the calling process's child processes terminate; it will then return a value of −1 with `errno` set to ECHILD.

**exit**  If in the exiting process's parent process the `func` value of SIGCLD is set to SIG_IGN, the exiting process will not create a zombie process.

When processing a pipeline, the shell makes the last process in the pipeline the parent of the proceeding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should take care not to set SIGCLD to be caught.

**NOTES**

SIGPOLL is issued when a file descriptor corresponding to a STREAMS [see intro(2)] file has a "selectable" event pending. A process must specifically request that this signal be sent using the I_SETSIG `ioctl` call [see streamio(7)]. Otherwise, the process will never receive SIGPOLL.

For portability, applications should use only the symbolic names of signals rather than their values and use only the set of signals defined here. The action for the signal SIGKILL can not be changed from the default system action.

Specific implementations may have other implementation-defined signals. Also, additional implementation-defined arguments may be passed to the signal-catch handler for hardware-generated signals. For certain hardware-generated signals, it may not be possible to resume execution at the point of interruption.

The signal type SIGSEGV is reserved for the condition that occurs on an invalid access to a data object. If an implementation can detect this condition, this signal type should be used.

The other signal management functions, signal(2) and pause(2), should not be used in conjunction with these routines for a particular signal type.
NAME
stat, fstat — get file status

SYNOPSIS
#include <sys/types.h>
#include <sys/stat.h>

int stat (path, buf)
char *path;
struct stat *buf;

int fstat (fildes, buf)
int fildes;
struct stat *buf;

DESCRIPTION
Path points to a path name naming a file. Read, write, or execute permission of
the named file is not required, but all directories listed in the path name leading
to the file must be searchable. stat obtains information about the named file.

Note that in a Remote File Sharing environment, the information returned by
stat depends upon the user/group mapping set up between the local and remote
computers. [See idload(1M)].

Fstat obtains information about an open file known by the file descriptor fildes,
obtained from a successful open, creat, dup, fcntl, or pipe system call.

Buf is a pointer to a stat structure into which information is placed concerning
the file.

The contents of the structure pointed to by buf include the following members:

ushort st_mode; /* File mode [see mknod(2)]*/
ino_t st_ino; /* Inode number */
dev_t st_dev; /* ID of device containing */
/* a directory entry for this file */
dev_t st_rdev; /* ID of device */
/* This entry is defined only for */
/* character special or block special files */
short st_nlink; /* Number of links */
ushort st_uid; /* User ID of the file's owner */
ushort st_gid; /* Group ID of the file's group */
off_t st_size; /* File size in bytes */
time_t st_atime; /* Time of last access */
time_t st_mtime; /* Time of last data modification */
time_t st_ctime; /* Time of last file status change */
/* Times measured in seconds since */
/* 00:00:00 GMT, Jan. 1, 1970 */

st_mode The mode of the file as described in the mknod(2) system call.

st_ino This field uniquely identifies the file in a given file system. The pair
st_ino and st_dev uniquely identifies regular files.

st_dev This field uniquely identifies the file system that contains the file. Its
value may be used as input to the ustat(2) system call to determine
more information about this file system. No other meaning is associated with this value.

- **st_rdev**: This field should be used only by administrative commands. It is valid only for block special or character special files and only has meaning on the system where the file was configured.

- **st_nlink**: This field should be used only by administrative commands.

- **st_uid**: The user ID of the file's owner.

- **st_gid**: The group ID of the file's group.

- **st_size**: For regular files, this is the address of the end of the file. For pipes or fifos, this is the count of the data currently in the file. For block special or character special, this is not defined.

- **st_atime**: Time when file data was last accessed. Changed by the following system calls: `creat(2)`, `mknod(2)`, `pipe(2)`, `utime(2)`, and `read(2)`.

- **st_mtime**: Time when data was last modified. Changed by the following system calls: `creat(2)`, `mknod(2)`, `pipe(2)`, `utime(2)`, and `write(2)`.

- **st_ctime**: Time when file status was last changed. Changed by the following system calls: `chmod(2)`, `chown(2)`, `creat(2)`, `link(2)`, `mknod(2)`, `pipe(2)`, `unlink(2)`, `utime(2)`, and `write(2)`.

`stat` will fail if one or more of the following are true:

- **ENOTDIR**: A component of the path prefix is not a directory.
- **ENOENT**: The named file does not exist.
- **EACCES**: Search permission is denied for a component of the path prefix.
- **EFAULT**: Buf or path points to an invalid address.
- **EINVAL**: A signal was caught during the `stat` system call.
- **ENOLINK**: Path points to a remote machine and the link to that machine is no longer active.
- **EMULTIHOP**: Components of path require hopping to multiple remote machines.

`Fstat` will fail if one or more of the following are true:

- **EBADF**: Fildes is not a valid open file descriptor.
- **EFAULT**: Buf points to an invalid address.
- **ENOLINK**: Fildes points to a remote machine and the link to that machine is no longer active.
SEE ALSO
chmod(2), chown(2), creat(2), link(2), mknod(2), pipe(2), read(2), time(2),
unlink(2), utime(2), write(2).

DIAGNOSTICS
Upon successful completion a value of 0 is returned. Otherwise, a value of −1
is returned and errno is set to indicate the error.
NAME
statfs, fstatfs — get file system information

SYNOPSIS
#include <sys/types.h>
#include <sys/statfs.h>

int statfs (path, buf, len, fstyp)
char *path;
struct statfs *buf;
int len, fstyp;

int fstatfs (fildes, buf, len, fstyp)
int fildes;
struct statfs *buf;
int len, fstyp;

DESCRIPTION
statfs returns a "generic superblock" describing a file system. It can be used to acquire information about mounted as well as unmounted file systems, and usage is slightly different in the two cases. In all cases, buf is a pointer to a structure (described below) which will be filled by the system call, and len is the number of bytes of information which the system should return in the structure. Len must be no greater than sizeof (struct statfs) and ordinarily it will contain exactly that value; if it holds a smaller value the system will fill the structure with that number of bytes. (This allows future versions of the system to grow the structure without invalidating older binary programs.)

If the file system of interest is currently mounted, path should name a file which resides on that file system. In this case the file system type is known to the operating system and the fstyp argument must be zero. For an unmounted file system path must name the block special file containing it and fstyp must contain the (non-zero) file system type. In both cases read, write, or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.

The statfs structure pointed to by buf includes the following members:

short  f fstyp;  /* File system type */
short  f bsize;  /* Block size */
short  f rsize;  /* Fragment size */
long   f blocks;  /* Total number of blocks */
long   f bfree;  /* Count of free blocks */
long   f files;  /* Total number of file nodes */
long   f ffree;  /* Count of free file nodes */
char   f fname[6];  /* Volume name */
char   f fpack[6];  /* Pack name */

fstatfs is similar, except that the file named by path in statfs is instead identified by an open file descriptor fildes obtained from a successful open(2), creat(2), dup(2), fcntl(2), or pipe(2) system call.

statfs obsoletes ustat(2) and should be used in preference to it in new programs.
statfs and fstatfs will fail if one or more of the following are true:

- [ENOTDIR] A component of the path prefix is not a directory.
- [ENOENT] The named file does not exist.
- [EACCES] Search permission is denied for a component of the path prefix.
- [EFAULT] Buf or path points to an invalid address.
- [EBADF] Fildes is not a valid open file descriptor.
- [EINVAL] Fstyp is an invalid file system type; path is not a block special file and fstyp is nonzero; len is negative or is greater than sizeof (struct statfs).
- [ENOLINK] Path points to a remote machine, and the link to that machine is no longer active.
- [EMULTIHOP] Components of path require hopping to multiple remote machines.

**DIAGNOSTICS**

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

**SEE ALSO**

chmod(2), chown(2), creat(2), link(2), mknod(2), pipe(2), read(2), time(2), unlink(2), utime(2), write(2), fs(4).
NAME
stime — set time

SYNOPSIS
int stime (tp)
long *tp;

DESCRIPTION
stime sets the system’s idea of the time and date. Tp points to the value of time as measured in seconds from 00:00:00 GMT January 1, 1970.

[EPERM] stime will fail if the effective user ID of the calling process is not super-user.

SEE ALSO
time(2).

DIAGNOSTICS
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
NAME
    sync — update super block

SYNOPSIS
    void sync ( )

DESCRIPTION
    *sync* causes all information in memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

    It should be used by programs which examine a file system, for example *fsck*, *df*, etc. It is mandatory before a re-boot.

    The writing, although scheduled, is not necessarily complete upon return from *sync*. 
NAME
sys3b — machine specific functions

SYNOPSIS
#include <sys/sys3b.h>
int sys3b (cmd, arg1, arg2, arg3)
int cmd, arg1, arg2, arg3;

DESCRIPTION
sys3b implements machine specific functions. The cmd argument determines the
function performed. The number of arguments expected is dependent on the
function.

Command GRNFLASH
When cmd is GRNFLASH, no arguments are expected. This function starts the
green LED flashing. Note that this command is only available to the super-user.

Command GRNON
When cmd is GRNON, no arguments are expected. This function turns the
green LED to a solid on state. Note that this command is only available to the
super-user.

Command RNVR
When cmd is RNVR, an argument of type struct nvparams * is expected.

struct nvparams{
    char *addr;
    char *data;
    unsigned short cnt;
}

This function reads cnt bytes at address addr in NVRAM into address data. Note
that this command is only available to the super-user.

Command RTODC
When cmd is RTODC, an argument of type struct tvdc * is expected.

struct tvdc {
    short htenths; short hsecs; short hmins;
    short hhours; short hdays; short hweekday;
    short hmonth; short hyear
}

This function reads the hardware time of day clock and returns the data in the
structure referred to by the argument. Note that this command is only available
to the super-user.

Command S3BSYM
When cmd is S3BSYM, the symbol table created during a self-config boot process
may be accessed. The symbols available via this command are defined in one of
two places: the driver routines loaded or the variable specifications in the files in
the /etc/master directory. Two arguments are expected; the first must be a
pointer to a buffer into which the symbol table is copied, and the second must
be an integer containing the total size of the buffer. The format of the symbol table is:

```c
int size;  /* symbol size in bytes */
int count; /* total number of symbols */
char name[]; /* for each symbol ... */
  /* name of symbol, padded with */
  '\0' to next sizeof(long) */
  /* boundary */
long value; /* value of symbol */
```

Typically, the symbol table would be retrieved with two calls to `sys3b`. First, the size of the symbol table is obtained by calling `sys3b` with a buffer of one integer. This integer is then used to obtain a buffer large enough to contain the entire symbol table. The second invocation of `sys3b` with this newly obtained buffer retrieves the entire symbol table.

```c
#include <sys/sys3b.h>

int size;  /* size of buffer needed */
struct s3bsym *buffer;  /* buffer pointer */

sys3b(S3BSYM, (struct s3bsym *) &size, sizeof(size));
buffer = (struct s3bsym *) malloc( size );
sys3b(S3BSYM, buffer, size);
```

Command S3BCONF

When `cmd` is S3BCONF, the configuration table created during a self-config boot process may be accessed. This table contains the names and locations of the devices supported by the currently running UNIX system, the names of all software modules included in the system, and the names of all devices in the EDT that were ignored. Two arguments are expected; the first must be a pointer to a buffer into which the configuration table is copied, and the second must be an integer containing the total size of the buffer. The format of the configuration table is:

```c
int ndev;  /* total number of entries */

/* for each entry ... */
long timestamp;  /* _f_timdat from file header */
char name[14];  /* name of device/module */
char flag;  /* configuration information */
  /* 0x80: device ignored */
  /* 0x40: name[] is a driver */
  /* 0x20: name[] is a software module */
char board;  /* local bus address of device */
```
Typically, the configuration table would be retrieved with two calls to \texttt{sys3b}. First, the number of entries is obtained by calling \texttt{sys3b} with a buffer of one integer. This integer is then used to calculate and obtain a buffer large enough to contain the entire configuration table. The second invocation of \texttt{sys3b} with this newly obtained buffer retrieves the configuration table.

```c
#include <sys/sys3b.h>

int count; /* total number of devices */
int size; /* size of buffer needed */
struct s3bconf *buffer; /* buffer pointer */
```

\texttt{sys3b(S3BCONF, (struct s3bconf *)&count, sizeof(count));}
size = sizeof(int);
size += count * sizeof(struct s3bc);
buffer = (struct s3bconf *)malloc(size);
\texttt{sys3b(S3BCONF, buffer, size);}\

Command S3BBOOT

When \texttt{cmd} is S3BBOOT, the timestamp and boot program path name used for a self-config boot process may be accessed. The path name of the a.out format file which was booted, and the timestamp from the file header [see \texttt{a.out(4)}] are saved. One argument is expected; a pointer to a buffer into which the information is copied. The format of this information is:

```c
long timestamp; /* f_timdat from file header */
char path[100]; /* path name */
```

This information would be retrieved with a single call to \texttt{sys3b}.

```c
#include <sys/sys3b.h>

struct s3bbboot buffer; /* buffer */
\texttt{sys3b(S3BBOOT, &buffer);}\
```

Command S3BAUTO

When \texttt{cmd} is S3BAUTO, no arguments are expected. This function returns a boolean value in answer to the question, "Was the last boot an auto-config boot, or was a fully configured file booted?" The value returned is zero if a fully configured file (such as /unix) was booted. The integer value 1 is returned if the preceding boot was an auto-config boot. The value is undefined if the system was booted in "magic mode." Note that this command is available only to the super-user.

Command S3BFPHW

When \texttt{cmd} is S3BFPHW, an indication of whether or not a MAU is present is returned. (See the \textit{Introduction} to this manual for a description of the MAU.) One argument, the address of an int, is expected. On return from the system call, this int will contain a 1 if a MAU is present or a 0 if a MAU is not present.
If the address of the int is not valid (i.e. not word aligned, nor user accessible, etc.) EFAULT will be returned.

To determine whether a MAU is present, the following should be done:

```
#include <sys/sys3b.h>

int mau_present;
sys3b(S3BFPHW, &mau_present);
```

If this command succeeds, it returns 0 to the calling process. This call will fail, returning -1, if one or more of the following is true:

[EFAULT] mau_present is not an integer,

[EFAULT] &mau_present is an invalid address.

**Command S3BSWPI**

Note: This cmd is available only with System V Releases 2.1 and 3.0 software.

When cmd is S3BSWPI, individual swapping areas may be added, deleted or the current areas determined. The address of an appropriately primed swap buffer is passed as the only argument. (Refer to sys/swap.h header file for details of loading the buffer.)

The format of the swap buffer is:

```
struct swapint {
    char si_cmd;  /*command: list, add, delete*/
    char *si_buf; /*swap file path pointer*/
    int si_swaplo; /*start block*/
    int si_nbblks; /*swap size*/
}
```

Note that the add and delete options of the command may only be exercised by the super-user.

Typically, a swap area is added by a single call to sys3b. First, the swap buffer is primed with appropriate entries for the structure members. Then sys3b is invoked.

```
#include <sys/sys3b.h>
#include <sys/swap.h>

struct swapint swapbuf; /*swap into buffer ptr*/

sys3b(S3BSWPI, &swapbuf);
```

If this command succeeds, it returns 0 to the calling process. This command fails, returning -1, if one or more of the following is true:

[EFAULT] Swapbuf points to an invalid address

[EFAULT] Swapbuf.si_buf points to an invalid address
[ENOTBLK] Swap area specified is not a block special device
[EXIST] Swap area specified has already been added
[ENOSPC] Too many swap areas in use (if adding)
[ENOMEM] Tried to delete last remaining swap area
[EINVAL] Bad arguments
[ENOMEM] No place to put swapped pages when deleting a swap area

Command STIME
When cmd is STIME, an argument of type long is expected. This function sets the system time and date. The argument contains the time as measured in seconds from 00:00:00 GMT January 1, 1970. Note that this command is only available to the super-user.

Command WNVR
When cmd is WNVR, an argument of type struct nvparams * is expected (see command RNVR). This function writes cnt bytes into address addr in NVRAM from address data. Note that this command is only available to the super-user.

SEE ALSO
sync(2), a.out(4).

DIAGNOSTICS
Upon successful completion, the value returned depends on cmd as follows:

    S3BSYM  A value of zero.
    S3BCONF A value of zero.
    S3BBOOT A value of zero.
    S3BAUTO A value of zero if a fully-configured file (such as /unix) was booted. A value of one if an auto-config boot was performed.

Otherwise, a value of -1 is returned and errno is set to indicate the error. When cmd is invalid, errno is set to EINVAL on return.
NAME
sysfs — get file system type information

SYNOPSIS
#include <sys/fstyp.h>
#include <sys/fsid.h>

int sysfs (opcode, fsname)
int opcode;
char *fsname;

int sysfs (opcode, fs_index, buf)
int opcode;
int fs_index;
char *buf;

int sysfs (opcode)
int opcode;

DESCRIPTION
sysfs returns information about the file system types configured in the system. The number of arguments accepted by sysfs varies and depends on the opcode. The currently recognized opcodes and their functions are described below:

GETFSIND translates fsname, a null-terminated file-system identifier, into a file-system type index.

GETFSTYP translates fs_index, a file-system type index, into a null-terminated file-system identifier and writes it into the buffer pointed to by buf; this buffer must be at least of size FSTYPSZ as defined in <sys/fstyp.h>.

GETNFSTYP returns the total number of file system types configured in the system.

sysfs will fail if one or more of the following are true:

[EINVAL] Fsname points to an invalid file-system identifier; fs_index is zero, or invalid; opcode is invalid.

[EFAULT] Buf or fsname point to an invalid user address.

DIAGNOSTICS
Upon successful completion, sysfs returns the file-system type index if the opcode is GETFSIND, a value of 0 if the opcode is GETFSTYP, or the number of file system types configured if the opcode is GETNFSTYP. Otherwise, a value of -1 is returned and errno is set to indicate the error.
NAME
time — get time

SYNOPSIS
#include <sys/types.h>

time_t time (tloc)
long *tloc;

DESCRIPTION
time returns the value of time in seconds since 00:00:00 GMT, January 1, 1970.
If tloc is non-zero, the return value is also stored in the location to which tloc points.

SEE ALSO
stime(2).

WARNING
time fails and its actions are undefined if tloc points to an illegal address.

DIAGNOSTICS
Upon successful completion, time returns the value of time. Otherwise, a value of −1 is returned and errno is set to indicate the error.
NAME
	times – get process and child process times

SYNOPSIS

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

long times (buffer)
struct tms *buffer;

DESCRIPTION

times fills the structure pointed to by buffer with time-accounting information.
The following are the contents of this structure:

struct tms {
  time_t tms_utime;
  time_t tms_stime;
  time_t tms_cutime;
  time_t tms_cstime;
};

This information comes from the calling process and each of its terminated child
processes for which it has executed a wait. All times are reported in clock ticks
per second. Clock ticks are a system-dependent parameter. The specific value
for an implementation is defined by the variable HZ, found in the include file
param.h.

Tms_utime is the CPU time used while executing instructions in the user space of
the calling process.

Tms_stime is the CPU time used by the system on behalf of the calling process.

Tms_cutime is the sum of the tms_utimes and tms_cutimes of the child processes.

Tms_cstime is the sum of the tms_stimes and tms_cstimes of the child processes.

[EFAULT] times will fail if buffer points to an illegal address.

SEE ALSO

eexec(2), fork(2), time(2), wait(2).

DIAGNOSTICS

Upon successful completion, times returns the elapsed real time, in clock ticks
per second, from an arbitrary point in the past (e.g., system start-up time). This
point does not change from one invocation of times to another. If times fails, a
-1 is returned and errno is set to indicate the error. On a 3B2 Computer clock
ticks occur 100 times per second.
NAME
uadmin — administrative control

SYNOPSIS
#include <sys/uadmin.h>
int uadmin (cmd, fcn, mdep)
int cmd, fcn, mdep;

DESCRIPTION
uadmin provides control for basic administrative functions. This system call is
tightly coupled to the system administrative procedures and is not intended for
general use. The argument mdep is provided for machine-dependent use and is
not defined here.

As specified by cmd, the following commands are available:

A_SHUTDOWN The system is shutdown. All user processes are killed, the
buffer cache is flushed, and the root file system is unmounted.
The action to be taken after the system has been shut down is
specified by fcn. The functions are generic; the hardware capa-
bilities vary on specific machines.

AD_HALT Halt the processor and turn off the power.

AD_BOOT Reboot the system, using /unix.

AD_IBOOT Interactive reboot; user is prompted for system
name.

A_REBOOT The system stops immediately without any further processing.
The action to be taken next is specified by fcn as above.

A_REMOUNT The root file system is mounted again after having been fixed.
This should be used only during the startup process.

uadmin fails if any of the following are true:

[EPERM] The effective user ID is not super-user.

DIAGNOSTICS
Upon successful completion, the value returned depends on cmd as follows:

A_SHUTDOWN Never returns.
A_REBOOT Never returns.
A_REMOUNT 0

Otherwise, a value of -1 is returned and errno is set to indicate the error.
NAME
ulimit — get and set user limits

SYNOPSIS

long ulimit (cmd, newlimit)
int cmd;
long newlimit;

DESCRIPTION
This function provides for control over process limits. The cmd values available are:

1  Get the regular file size limit of the process. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read.

2  Set the regular file size limit of the process to the value of newlimit. Any process may decrease this limit, but only a process with an effective user ID of super-user may increase the limit. ulimit fails and the limit is unchanged if a process with an effective user ID other than super-user attempts to increase its regular file size limit. [EPERM]

3  Get the maximum possible break value [see brk(2)].

SEE ALSO
brk(2), write(2).

WARNING
ulimit is effective in limiting the growth of regular files. Pipes are currently limited to 5,120 bytes.

DIAGNOSTICS
Upon successful completion, a non-negative value is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.
NAME

umask — set and get file creation mask

SYNOPSIS

```
int umask (cmask)
int cmask;
```

DESCRIPTION

`umask` sets the process's file mode creation mask to `cmask` and returns the previous value of the mask. Only the low-order 9 bits of `cmask` and the file mode creation mask are used.

SEE ALSO

`chmod(2), creat(2), mknod(2), open(2),

DIAGNOSTICS

The previous value of the file mode creation mask is returned.
umount - unmount a file system

int umount (file)
char *file;

DESCRIPTION
umount requests that a previously mounted file system contained on the block
special device or directory identified by file be unmounted. File is a pointer to a
path name. After unmounting the file system, the directory upon which the file
system was mounted reverts to its ordinary interpretation.

umount may be invoked only by the super-user.

umount will fail if one or more of the following are true:

[EPERM] The process's effective user ID is not super-user.
[EINVAL] File does not exist.
[ENOTBLK] File is not a block special device.
[EINVAL] File is not mounted.
[EBUSY] A file on file is busy.
[EFAULT] File points to an illegal address.
[EREMOTE] File is remote.
[ENOLINK] File is on a remote machine, and the link to that machine is no
longer active.
[EMULTIHOP] Components of the path pointed to by file require hopping to
multiple remote machines.

SEE ALSO
mount(2).

DIAGNOSTICS
Upon successful completion a value of 0 is returned. Otherwise, a value of -1
is returned and errno is set to indicate the error.
NAME
uname — get name of current UNIX system

SYNOPSIS
#include <sys/utsname.h>
int uname (name)
struct utsname *name;

DESCRIPTION
uname stores information identifying the current UNIX system in the structure
pointed to by name.
uname uses the structure defined in <sys/utsname.h> whose members are:

char sysname[9];
char nodename[9];
char release[9];
char version[9];
char machine[9];

uname returns a null-terminated character string naming the current UNIX system
in the character array sysname. Similarly, nodename contains the name that the
system is known by on a communications network. Release and version further
identify the operating system. Machine contains a standard name that identifies
the hardware that the UNIX system is running on.

EFAULT] uname will fail if name points to an invalid address.

SEE ALSO

DIAGNOSTICS
Upon successful completion, a non-negative value is returned. Otherwise, a
value of −1 is returned and errno is set to indicate the error.
UNLINK(2)

NAME
unlink — remove directory entry

SYNOPSIS
    int unlink (path)
    char *path;

DESCRIPTION
unlink removes the directory entry named by the path name pointed to by path.
The named file is unlinked unless one or more of the following are true:

[ENOTDIR]   A component of the path prefix is not a directory.
[ENOENT]    The named file does not exist.
[EACCES]    Search permission is denied for a component of the path prefix.
[EACCES]    Write permission is denied on the directory containing the link to be removed.
[EPERM]     The named file is a directory and the effective user ID of the process is not super-user.
[EBUSY]     The entry to be unlinked is the mount point for a mounted file system.
[ETXTBSY]   The entry to be unlinked is the last link to a pure procedure (shared text) file that is being executed.
[EROFS]     The directory entry to be unlinked is part of a read-only file system.
[EFAULT]   Path points outside the process’s allocated address space.
[EINTR]     A signal was caught during the unlink system call.
[ENOLINK]   Path points to a remote machine and the link to that machine is no longer active.
[EMULTIHOP] Components of path require hopping to multiple remote machines.

When all links to a file have been removed and no process has the file open, the space occupied by the file is freed and the file ceases to exist. If one or more processes have the file open when the last link is removed, the removal is postponed until all references to the file have been closed.

SEE ALSO
    close(2), link(2), open(2).

DIAGNOSTICS
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.
NAME
ustat – get file system statistics

SYNOPSIS
#include <sys/types.h>
#include <ustat.h>

int ustat (dev, buf)
dev_t dev;
struct ustat *buf;

DESCRIPTION
ustat returns information about a mounted file system. Dev is a device number identifying a device containing a mounted file system. Buf is a pointer to a ustat structure that includes the following elements:

- daddr_t f_free; /* Total free blocks */
- ino_t f_inode; /* Number of free inodes */
- char f_name[6]; /* Filsys name */
- char f_pack[6]; /* Filsys pack name */

ustat will fail if one or more of the following are true:

- EINVAL  Dev is not the device number of a device containing a mounted file system.
-EFAULT  Buf points outside the process’s allocated address space.
-ENINTR  A signal was caught during a ustat system call.
-ENOLINK  Dev is on a remote machine and the link to that machine is no longer active.
-ECOMM  Dev is on a remote machine and the link to that machine is no longer active.

SEE ALSO
stat(2), fs(4).

DIAGNOSTICS
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.
NAME
utime – set file access and modification times

SYNOPSIS
#include <sys/types.h>
int utime (path, times)
char *path;
struct utimbuf *times;

DESCRIPTION
Path points to a path name naming a file. utime sets the access and modification
times of the named file.

If times is NULL, the access and modification times of the file are set to the
current time. A process must be the owner of the file or have write permission
to use utime in this manner.

If times is not NULL, times is interpreted as a pointer to a utimbuf structure and
the access and modification times are set to the values contained in the desig­
nated structure. Only the owner of the file or the super-user may use utime this
way.

The times in the following structure are measured in seconds since 00:00:00
GMT, Jan. 1, 1970.

struct utimbuf {
    time_t actime; /* access time */
    time_t modtime; /* modification time */
};

utime will fail if one or more of the following are true:
[ENOENT] The named file does not exist.
[ENOTDIR] A component of the path prefix is not a directory.
[EACCES] Search permission is denied by a component of the path prefix.
[EPERM] The effective user ID is not super-user and not the owner of the
file and times is not NULL.
[EACCES] The effective user ID is not super-user and not the owner of the
file and times is NULL and write access is denied.
[ELOSS] The file system containing the file is mounted read-only.
[EFAULT] Times is not NULL and points outside the process’s allocated
address space.
[EFAULT] Path points outside the process’s allocated address space.
[EINTR] A signal was caught during the utime system call.
[ENOLINK] Path points to a remote machine and the link to that machine is
no longer active.
[EMULTIHOP] Components of path require hopping to multiple remote
machines.

SEE ALSO
stat(2).
DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.
NAME
wait — wait for child process to stop or terminate

SYNOPSIS
```c
int wait (stat_loc)
int *stat_loc;
```

DESCRIPTION
`wait` suspends the calling process until until one of the immediate children terminates or until a child that is being traced stops, because it has hit a break point. The `wait` system call will return prematurely if a signal is received and if a child process stopped or terminated prior to the call on `wait`, return is immediate.

If `stat_loc` is non-zero, 16 bits of information called status are stored in the low order 16 bits of the location pointed to by `stat_loc`. Status can be used to differentiate between stopped and terminated child processes and if the child process terminated, status identifies the cause of termination and passes useful information to the parent. This is accomplished in the following manner:

If the child process stopped, the high order 8 bits of status will contain the number of the signal that caused the process to stop and the low order 8 bits will be set equal to 0177.

If the child process terminated due to an `exit` call, the low order 8 bits of status will be zero and the high order 8 bits will contain the low order 8 bits of the argument that the child process passed to `exit` [see `exit(2)`].

If the child process terminated due to a signal, the high order 8 bits of status will be zero and the low order 8 bits will contain the number of the signal that caused the termination. In addition, if the low order seventh bit (i.e., bit 200) is set, a "core image" will have been produced [see `signal(2)`].

If a parent process terminates without waiting for its child processes to terminate, the parent process ID of each child process is set to 1. This means the initialization process inherits the child processes [see `intro(2)`].

`wait` will fail and return immediately if one or more of the following are true:

[ECHILD] The calling process has no existing unwaited-for child processes.

SEE ALSO
`exec(2)`, `exit(2)`, `fork(2)`, `intro(2)`, `pause(2)`, `ptrace(2)`, `signal(2)`.

WARNING
`wait` fails and its actions are undefined if `stat_loc` points to an invalid address.

See WARNING in `signal(2)`.
WAIT(2)

DIAGNOSTICS

If `wait` returns due to the receipt of a signal, a value of −1 is returned to the calling process and `errno` is set to EINTR. If `wait` returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of −1 is returned and `errno` is set to indicate the error.
NAME
write — write on a file

SYNOPSIS
int write (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;

DESCRIPTION
fildes is a file descriptor obtained from a creat(2), open(2), dup(2), fcntl(2), or
pipe(2) system call.
write attempts to write nbyte bytes from the buffer pointed to by buf to the file
associated with the fildes.

On devices capable of seeking, the actual writing of data proceeds from the posi­
tion in the file indicated by the file pointer. Upon return from write, the file
pointer is incremented by the number of bytes actually written.

On devices incapable of seeking, writing always takes place starting at the
current position. The value of a file pointer associated with such a device is
undefined.

If the O_APPEND flag of the file status flags is set, the file pointer will be set to
the end of the file prior to each write.

For regular files, if the O_SYNC flag of the file status flags is set, the write will
not return until both the file data and file status have been physically updated.
This function is for special applications that require extra reliability at the cost of
performance. For block special files, if O_SYNC is set, the write will not return
until the data has been physically updated.

A write to a regular file will be blocked if mandatory file/record locking is set
[see chmod(2)], and there is a record lock owned by another process on the seg­
ment of the file to be written. If O_NDELAY is not set, the write will sleep until
the blocking record lock is removed.

For STREAMS [see intro(2)] files, the operation of write is determined by the
values of the minimum and maximum nbyte range ("packet size") accepted by
the stream. These values are contained in the topmost stream module. Unless
the user pushes [see I PUSH in streamio(7)] the topmost module, these values can
not be set or tested from user level. If nbyte falls within the packet size range,
nbyte bytes will be written. If nbyte does not fall within the range and the
minimum packet size value is zero, write will break the buffer into maximum
packet size segments prior to sending the data downstream (the last segment
may contain less than the maximum packet size). If nbyte does not fall within
the range and the minimum value is non-zero, write will fail with errno set to
ERANGE. Writing a zero-length buffer (nbyte is zero) sends zero bytes with zero
returned.

For STREAMS files, if O_NDELAY is not set and the stream can not accept data
(the stream write queue is full due to internal flow control conditions), write will
block until data can be accepted. O_NDELAY will prevent a process from
blocking due to flow control conditions. If O_NDELAY is set and the stream can
not accept data, write will fail. If O_NDELAY is set and part of the buffer has
been written when a condition in which the stream can not accept additional
data occurs, write will terminate and return the number of bytes written.

write will fail and the file pointer will remain unchanged if one or more of the
following are true:

- **[EAGAIN]** Mandatory file/record locking was set, O_NDELAY was set, and
  there was a blocking record lock.
- **[EAGAIN]** Total amount of system memory available when reading via
  raw IO is temporarily insufficient.
- **[EAGAIN]** Attempt to write to a stream that can not accept data with the
  O_NDELAY flag set.
- **[EBADF]** fildes is not a valid file descriptor open for writing.
- **[EDEADLK]** The write was going to go to sleep and cause a deadlock situa-
  tion to occur.
- **[EFAULT]** buf points outside the process’s allocated address space.
- **[EFBIG]** An attempt was made to write a file that exceeds the process’s
  file size limit or the maximum file size [see ulimit(2)].
- **[EINVAL]** A signal was caught during the write system call.
- **[EINTR]** Attempt to write to a stream linked below a multiplexor.
- **[ENOLCK]** The system record lock table was full, so the write could not go
  to sleep until the blocking record lock was removed.
- **[ENOLINK]** fildes is on a remote machine and the link to that machine is no
  longer active.
- **[ENOSPC]** During a write to an ordinary file, there is no free space left on
  the device.
- **[ENXIO]** A hangup occurred on the stream being written to.
- **[EPIPE and SIGPIPE signal]** An attempt is made to write to a pipe that is not open for
  reading by any process.
- **[ERANGE]** Attempt to write to a stream with nbyte outside specified
  minimum and maximum write range, and the minimum value is non-zero.

If a write requests that more bytes be written than there is room for (e.g., the
ulimit [see ulimit(2)] or the physical end of a medium), only as many bytes as
there is room for will be written. For example, suppose there is space for 20
bytes more in a file before reaching a limit. A write of 512-bytes will return 20.
The next write of a non-zero number of bytes will give a failure return (except as
noted below).

If the file being written is a pipe (or FIFO) and the O_NDELAY flag of the file flag
word is set, then write to a full pipe (or FIFO) will return a count of 0. Other-
wise (O_NDELAY clear), writes to a full pipe (or FIFO) will block until space
becomes available.
A write to a STREAMS file can fail if an error message has been received at the stream head. In this case, \textit{errno} is set to the value included in the error message.

\textbf{SEE ALSO}
\begin{itemize}
\item creat(2), dup(2), fcntl(2), intro(2), lseek(2), open(2), pipe(2), ulimit(2).
\end{itemize}

\textbf{DIAGNOSTICS}
Upon successful completion the number of bytes actually written is returned. Otherwise, \texttt{-1} is returned and \textit{errno} is set to indicate the error.
NAME
intro – introduction to functions and libraries

DESCRIPTION
This section describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume. Certain major collections are identified by a letter after the section number:

(3C) These functions, together with those of Section 2 and those marked (3S), constitute the Standard C Library libc, which is automatically loaded by the C compiler, cc(1). (For this reason the (3C) and (3S) sections together comprise one section of this manual.) The link editor ld(1) searches this library under the -lc option. A "shared library" version of libc can be searched using the -lc_s option, resulting in smaller a.outs. Declarations for some of these functions may be obtained from #include files indicated on the appropriate pages.

(3S) These functions constitute the "standard I/O package" [see stdio(3S)]. These functions are in the library libc, already mentioned. Declarations for these functions may be obtained from the #include file <stdio.h>.

(3M) These functions constitute the Math Library, libm. They are automatically loaded as needed by the FORTRAN compiler f77(1). They are not automatically loaded by the C compiler, cc(1); however, the link editor searches this library under the -lm option. Declarations for these functions may be obtained from the #include file <math.h>. Several generally useful mathematical constants are also defined there [see math(5)].

(3N) This contains sets of functions constituting the Network Services library. These sets provide protocol independent interfaces to networking services based on the service definitions of the OSI (Open Systems Interconnection) reference model. Application developers access the function sets that provide services at a particular level. The function sets contained in the library are:

- TRANSPORT INTERFACE (TI) - provide the services of the OSI Transport Layer. These services provide reliable end-to-end data transmission using the services of an underlying network. Applications written using the TI functions are independent of the underlying protocols. Declarations for these functions may be obtained from the #include file <tiuser.h>. The link editor ld(1) searches this library under the -lnsl_s option.

(3X) Various specialized libraries. The files in which these libraries are found are given on the appropriate pages.

(3F) These functions constitute the FORTRAN intrinsic function library, libF77. These functions are automatically available to the FORTRAN programmer and require no special invocation of the compiler.

DEFINITIONS
A character is any bit pattern able to fit into a byte on the machine. The null character is a character with value 0, represented in the C language as '\0'. A character array is a sequence of characters. A null-terminated character array is a sequence of characters, the last of which is the null character. A string is a
designation for a null-terminated character array. The null string is a character array containing only the null character. A NULL pointer is the value that is obtained by casting 0 into a pointer. The C language guarantees that this value will not match that of any legitimate pointer, so many functions that return pointers return it to indicate an error. NULL is defined as 0 in <stdio.h>; the user can include an appropriate definition if not using <stdio.h>.

Many groups of FORTRAN intrinsic functions have generic function names that do not require explicit or implicit type declaration. The type of the function will be determined by the type of its argument(s). For example, the generic function max will return an integer value if given integer arguments (maxO), a real value if given real arguments (amaxl), or a double-precision value if given double-precision arguments (dmaxl).

**Netbuf** In the Network Services library, netbuf is a structure used in various Transport Interface (TI) functions to send and receive data and information. It contains the following members:

```c
unsigned int maxlen;
unsigned int len;
char *buf;
```

*Buf* points to a user input and/or output buffer. *Len* generally specifies the number of bytes contained in the buffer. If the structure is used for both input and output, the function will replace the user value of *len* on return.

*Maxlen* generally has significance only when *buf* is used to receive output from the TI function. In this case, it specifies the physical size of the buffer, the maximum value of *len* that can be set by the function. If *maxlen* is not large enough to hold the returned information, an TBUVOFLW error will generally result. However, certain functions may return part of the data and not generate an error.

**FILES**

LIBDIR usually /lib
LIBDIR/libc.a
LIBDIR/libc_s.a
LIBDIR/libm.a
LIBDIR/lib77.a
/shlib/libc_s
/shlib/libnsl_s (3N)
/usr/lib/libnsl_s.a (3N)

**SEE ALSO**

ar(l), cc(l), ld(l), lint(l), intro(2), stdio(3S), math(5).
f77(1) in the FORTRAN Programming Language Manual.

**DIAGNOSTICS**

Functions in the C and Math Libraries (3C and 3M) may return the conventional values 0 or ±HUGE (the largest-magnitude single-precision floating-point numbers; HUGE is defined in the <math.h> header file) when the function is
undefined for the given arguments or when the value is not representable. In these cases, the external variable *errno* [see intro(2)] is set to the value EDOM or ERANGE.

**WARNING**

Many of the functions in the libraries call and/or refer to other functions and external variables described in this section and in Section 2 (System Calls). If a program inadvertently defines a function or external variable with the same name, the presumed library version of the function or external variable may not be loaded. The *lint*(1) program checker reports name conflicts of this kind as "multiple declarations" of the names in question. Definitions for Sections 2, 3C, and 3S are checked automatically. Other definitions can be included by using the –I option. (For example, –Im includes definitions for Section 3M, the Math Library.) Use of *lint* is highly recommended.
NAME
  a64l, l64a — convert between long integer and base-64 ASCII string

SYNOPSIS
  long a64l (s)
  char *s;
  char *l64a (l)
  long l;

DESCRIPTION
  These functions are used to maintain numbers stored in base-64 ASCII characters. This is a notation by which long integers can be represented by up to six characters; each character represents a "digit" in a radix-64 notation.

  The characters used to represent "digits" are . for 0, / for 1, 0 through 9 for 2–11, A through Z for 12–37, and a through z for 38–63.

  a64l takes a pointer to a null-terminated base-64 representation and returns a corresponding long value. If the string pointed to by s contains more than six characters, a64l will use the first six.

  a64l scans the character string from left to right, decoding each character as a 6 bit Radix 64 number.

  l64a takes a long argument and returns a pointer to the corresponding base-64 representation. If the argument is 0, l64a returns a pointer to a null string.

CAVEAT
  The value returned by l64a is a pointer into a static buffer, the contents of which are overwritten by each call.
NAME
abort — generate an IOT fault

SYNOPSIS
int abort ( )

DESCRIPTION
abort does the work of exit(2), but instead of just exiting, abort causes SIGABRT
to be sent to the calling process. If SIGABRT is neither caught nor ignored, all
stdio(3S) streams are flushed prior to the signal being sent, and a core dump
results.

abort returns the value of the kill(2) system call.

SEE ALSO
sdb(1), exit(2), kill(2), signal(2).

DIAGNOSTICS
If SIGABRT is neither caught nor ignored, and the current directory is writable, a
core dump is produced and the message “abort — core dumped” is written by
the shell.
NAME
   abs — return integer absolute value

SYNOPSIS
   int abs (i)
   int i;

DESCRIPTION
   abs returns the absolute value of its integer operand.

SEE ALSO
   floor(3M).

CAVEAT
   In two’s-complement representation, the absolute value of the negative integer
   with largest magnitude is undefined. Some implementations trap this error, but
   others simply ignore it.
NAME
bsearch — binary search a sorted table

SYNOPSIS
#include <search.h>
char *bsearch ((char *) key, (char *) base, nel, sizeof (*key), compar)
unsigned nel;
int (*compar)();

DESCRIPTION
bsearch is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It
returns a pointer into a table indicating where a datum may be found. The table
must be previously sorted in increasing order according to a provided com­
parison function. Key points to a datum instance to be sought in the table. Base
points to the element at the base of the table. Nel is the number of elements in
the table. Compar is the name of the comparison function, which is called with
two arguments that point to the elements being compared. The function must
return an integer less than, equal to, or greater than zero as accordingly the first
argument is to be considered less than, equal to, or greater than the second.

EXAMPLE
The example below searches a table containing pointers to nodes consisting of a
string and its length. The table is ordered alphabetically on the string in the
node pointed to by each entry.

This code fragment reads in strings and either finds the corresponding node and
prints out the string and its length, or prints an error message.
#include <stdio.h>
#include <search.h>

#define TABSIZE 1000

struct node {
    char *string;
    int length;
};
struct node table[TABSIZE]; /* table to be searched */

struct node *node_ptr, node;
int node_compare(); /* routine to compare 2 nodes */
char str_space[20]; /* space to read string into */
node.string = str_space;
while (scanf("%s", node.string) != EOF) {
    node_ptr = (struct node *)bsearch((char *)(&node),
          (char *)table, TABSIZE,
          sizeof(struct node), node_compare);
    if (node_ptr != NULL) {
        (void)printf("string = %20s, length = %d\n",
                     node_ptr->string, node_ptr->length);
    } else {
        (void)printf("not found: %s\n", node.string);
    }
}

This routine compares two nodes based on an alphabetical ordering of the string field.

int
node_compare(node1, node2)
char *node1, *node2;
{
    return (strcmp((struct node *)node1)->string,
                 (struct node *)node2)->string);
}

NOTES
The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.
The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.
Although bsearch is declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

SEE ALSO
hsearch(3C), lsearch(3C), qsort(3C), tsearch(3C).

DIAGNOSTICS
A NULL pointer is returned if the key cannot be found in the table.
NAME
clock — report CPU time used

SYNOPSIS
long clock ( )

DESCRIPTION
clock returns the amount of CPU time (in microseconds) used since the first call to clock. The time reported is the sum of the user and system times of the calling process and its terminated child processes for which it has executed wait(2), pclose(3S), or system(3S).
The resolution of the clock is 10 milliseconds on AT&T 3B computers.

SEE ALSO
times(2), wait(2), popen(3S), system(3S).

BUGS
The value returned by clock is defined in microseconds for compatibility with systems that have CPU clocks with much higher resolution. Because of this, the value returned will wrap around after accumulating only 2147 seconds of CPU time (about 36 minutes).
NAME
conv: toupper, tolower, _toupper, _tolower, toascii — translate characters

SYNOPSIS
#include <ctype.h>
int toupper (c)
int tolower (c)
int _toupper (c)
int _tolower (c)
int toascii (c)

DESCRIPTION
Toupper and tolower have as domain the range of getc(3S): the integers from -1 through 255. If the argument of toupper represents a lower-case letter, the result is the corresponding upper-case letter. If the argument of tolower represents an upper-case letter, the result is the corresponding lower-case letter. All other arguments in the domain are returned unchanged.

The macros _toupper and _tolower, are macros that accomplish the same thing as toupper and tolower but have restricted domains and are faster. _toupper requires a lower-case letter as its argument; its result is the corresponding upper-case letter. The macro _tolower requires an upper-case letter as its argument; its result is the corresponding lower-case letter. Arguments outside the domain cause undefined results.

Toascii yields its argument with all bits turned off that are not part of a standard ASCII character; it is intended for compatibility with other systems.

SEE ALSO
cctype(3C), getc(3S).
NAME
crypt, setkey, encrypt — generate hashing encryption

SYNOPSIS
char *crypt (key, salt)
char *key, *salt;
void setkey (key)
char *key;
void encrypt (block, ignored)
char *block;
int ignored;

DESCRIPTION
crypt is the password encryption function. It is based on a one way hashing
encryption algorithm with variations intended (among other things) to frustrate
use of hardware implementations of a key search.

Key is a user's typed password. Salt is a two-character string chosen from the
set [a-zA-Z0-9./]; this string is used to perturb the hashing algorithm in one of
4096 different ways, after which the password is used as the key to encrypt
repeatedly a constant string. The returned value points to the encrypted pass-
word. The first two characters are the salt itself.

The setkey and encrypt entries provide (rather primitive) access to the actual
hashing algorithm. The argument of setkey is a character array of length 64 con-
taining only the characters with numerical value 0 and 1. If this string is
divided into groups of 8, the low-order bit in each group is ignored; this gives a
56-bit key which is set into the machine. This is the key that will be used with
the hashing algorithm to encrypt the string block with the function encrypt.

The argument to the encrypt entry is a character array of length 64 containing
only the characters with numerical value 0 and 1. The argument array is
modified in place to a similar array representing the bits of the argument after
having been subjected to the hashing algorithm using the key set by setkey.
Ignored is unused by encrypt but it must be present.

SEE ALSO
getpass(3C), passwd(4),

CAVEAT
The return value points to static data that are overwritten by each call.
NAME
ctermid — generate file name for terminal

SYNOPSIS
#include <stdio.h>
char *ctermid (s)
char *s;

DESCRIPTION
ctermid generates the path name of the controlling terminal for the current process, and stores it in a string.

If s is a NULL pointer, the string is stored in an internal static area, the contents of which are overwritten at the next call to ctermid, and the address of which is returned. Otherwise, s is assumed to point to a character array of at least L_ctermid elements; the path name is placed in this array and the value of s is returned. The constant L_ctermid is defined in the <stdio.h> header file.

NOTES
The difference between ctermid and ttyname(3C) is that ttyname must be handed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while ctermid returns a string (/dev/tty) that will refer to the terminal if used as a file name. Thus ttyname is useful only if the process already has at least one file open to a terminal.

SEE ALSO
ttyname(3C).
NAME
cime, localtime, gmtime, asctime, tzset — convert date and time to string

SYNOPSIS
#include <sys/types.h>
#include <time.h>
char *ctime (clock)
time_t *clock;
struct tm *localtime (clock)
time_t *clock;
struct tm *gmtime (clock)
time_t *clock;
char *asctime (tm)
struct tm *tm;
extern long timezone;
extern int daylight;
extern char *tzname[2];
void tzset ()

DESCRIPTION
cime converts a long integer, pointed to by clock, representing the time in
seconds since 00:00:00 GMT, January 1, 1970, and returns a pointer to a 26-
character string in the following form. All the fields have constant width.

Sun Sep 16 01:03:52 1985

Localtime and gmtime return pointers to "tm" structures, described below. Local-
time corrects for the time zone and possible Daylight Savings Time; gmtime con-
verts directly to Greenwich Mean Time (GMT), which is the time the UNIX
system uses.

Asctime converts a "tm" structure to a 26-character string, as shown in the
above example, and returns a pointer to the string.

Declarations of all the functions and externals, and the "tm" structure, are in the
<time.h> header file. The structure declaration is:

struct tm {
    int tm_sec; /* seconds (0 - 59) */
    int tm_min; /* minutes (0 - 59) */
    int tm_hour; /* hours (0 - 23) */
    int tm_mday; /* day of month (1 - 31) */
    int tm_mon; /* month of year (0 - 11) */
    int tm_year; /* year - 1900 */
    int tm_wday; /* day of week (Sunday = 0) */
    int tm_yday; /* day of year (0 - 365) */
    int tm_isdst;
};

Tm_isdst is non-zero if Daylight Savings Time is in effect.
The external long variable `timezone` contains the difference, in seconds, between GMT and local standard time (in EST, `timezone` is 5\*60\*60); the external variable `daylight` is non-zero if and only if the standard U.S.A. Daylight Savings Time conversion should be applied. The program knows about the peculiarities of this conversion in 1974 and 1975; if necessary, a table for these years can be extended.

If an environment variable named `TZ` is present, `asctime` uses the contents of the variable to override the default time zone. The value of `TZ` must be a three-letter time zone name, followed by a number representing the difference between local time and Greenwich Mean Time in hours, followed by an optional three-letter name for a daylight time zone. For example, the setting for New Jersey would be `EST5EDT`. The effects of setting `TZ` are thus to change the values of the external variables `timezone` and `daylight`; in addition, the time zone names contained in the external variable

```
char *tzname[2] = { "EST", "EDT" };
```

are set from the environment variable `TZ`. The function `tzset` sets these external variables from `TZ`; `tzset` is called by `asctime` and may also be called explicitly by the user.

Note that in most installations, `TZ` is set by default when the user logs on, to a value in the local /etc/profile file [see `profile(4)`].

SEE ALSO

`time(2)`, `getenv(3C)`, `profile(4)`, `environ(5)`.

CAVEAT

The return values point to static data whose content is overwritten by each call.
NAME
cctype: isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscrtl, isascii — classify characters

SYNOPSIS
#include <ctype.h>
int isalpha (c)
int c;
...

DESCRIPTION
These macros classify character-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. Isascii is defined on all integer values; the rest are defined only where isascii is true and on the single non-ASCII value EOF [-1; see stdio(3S)].
isalpha c is a letter.
isupper c is an upper-case letter.
islower c is a lower-case letter.
isdigit c is a digit [0-9].
isxdigit c is a hexadecimal digit [0-9], [A-F] or [a-f].
isalnum c is an alphanumeric (letter or digit).
isspace c is a space, tab, carriage return, newline, vertical tab, or formfeed.
ispunct c is a punctuation character (neither control nor alphanumeric).
isprint c is a printing character, code 040 (space) through 0176 (tilde).
isgraph c is a printing character, like isprint except false for space.
iscrtl c is a delete character (0177) or an ordinary control character (less than 040).
isascii c is an ASCII character, code less than 0200.

SEE ALSO
stdio(3S), ascii(5).

DIAGNOSTICS
If the argument to any of these macros is not in the domain of the function, the result is undefined.
NAME
cuserid — get character login name of the user

SYNOPSIS
#include <stdio.h>
char *cuserid (s)
char *s;

DESCRIPTION
cuserid generates a character-string representation of the login name that the
owner of the current process is logged in under. If s is a NULL pointer, this
representation is generated in an internal static area, the address of which is
returned. Otherwise, s is assumed to point to an array of at least L_cuserid
characters; the representation is left in this array. The constant L_cuserid is
defined in the <stdio.h> header file.

DIAGNOSTICS
If the login name cannot be found, cuserid returns a NULL pointer; if s is not a
NULL pointer, a null character (\0) will be placed at s[0].

SEE ALSO
getlogin(3C), getpwent(3C).
NAME
dial — establish an out-going terminal line connection

SYNOPSIS
#include <dial.h>
int dial (call)
CALL call;
void undial (fd)
int fd;

DESCRIPTION
dial returns a file-descriptor for a terminal line open for read/write. The argu­ment to dial is a CALL structure (defined in the <dial.h> header file).

When finished with the terminal line, the calling program must invoke undial to
release the semaphore that has been set during the allocation of the terminal
device.

The definition of CALL in the <dial.h> header file is:

typedef struct {
    struct termio *attr;       /* pointer to termio attribute struct */
    int baud;                   /* transmission data rate */
    int speed;                  /* 212A modem: low=300, high=1200 */
    char *line;                 /* device name for out-going line */
    char *telno;                /* pointer to tel-no digits string */
    int modem;                  /* specify modem control for direct lines */
    char *device;               /*Will hold the name of the device used 
                                 to make a connection */
    int dev_len;                /* The length of the device used to make 
                                 connection */
} CALL;

The CALL element speed is intended only for use with an outgoing dialed call, in
which case its value should be either 300 or 1200 to identify the 113A modem,
or the high- or low-speed setting on the 212A modem. Note that the 113A
modem or the low-speed setting of the 212A modem will transmit at any rate
between 0 and 300 bits per second. However, the high-speed setting of the
212A modem transmits and receives at 1200 bits per second only. The CALL
element baud is for the desired transmission baud rate. For example, one might
set baud to 110 and speed to 300 (or 1200). However, if speed set to 1200 baud
must be set to high (1200).

If the desired terminal line is a direct line, a string pointer to its device-name
should be placed in the line element in the CALL structure. Legal values for
such terminal device names are kept in the L-devices file. In this case, the value
of the baud element need not be specified as it will be determined from the L-
devices file.
The `telno` element is for a pointer to a character string representing the telephone number to be dialed. Such numbers may consist only of symbols described on the `acu(7)`. The termination symbol will be supplied by the `dial` function, and should not be included in the `telno` string passed to `dial` in the CALL structure.

The CALL element `modem` is used to specify modem control for direct lines. This element should be non-zero if modem control is required. The CALL element `attr` is a pointer to a `termio` structure, as defined in the `termio.h` header file. A NULL value for this pointer element may be passed to the `dial` function, but if such a structure is included, the elements specified in it will be set for the outgoing terminal line before the connection is established. This is often important for certain attributes such as parity and baud-rate.

The CALL element `device` is used to hold the device name (cul..) that establishes the connection.

The CALL element `dev_len` is the length of the device name that is copied into the array `device`.

**FILES**

```
/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..itty-device
```

**SEE ALSO**

`alarm(2), read(2), write(2)`, `acu(7), termio(7)` in the *System Administrator’s Reference Manual*. `uucp(1C)` in the *User’s Reference Manual*.

**DIAGNOSTICS**

On failure, a negative value indicating the reason for the failure will be returned. Mnemonics for these negative indices as listed here are defined in the `<dial.h>` header file.

```
INTRPT    -1  /* interrupt occurred */
D_HUNG    -2  /* dialer hung (no return from write) */
NO_ANS     -3  /* no answer within 10 seconds */
ILL_BD     -4  /* illegal baud-rate */
A_PROB    -5  /* acu problem (open() failure) */
L_PROB    -6  /* line problem (open() failure) */
NO_Ldv    -7  /* can’t open LDEVS file */
DV_NT_A    -8  /* requested device not available */
DV_NT_K    -9  /* requested device not known */
NO_BD_A   -10  /* no device available at requested baud */
NO_BD_K   -11  /* no device known at requested baud */
```

**WARNINGS**

The `dial (3C)` library function is not compatible with Basic Networking Utilities on UNIX System V Release 2.0.

Including the `<dial.h>` header file automatically includes the `<termio.h>` header file.

The above routine uses `<stdio.h>`, which causes it to increase the size of programs, not otherwise using standard I/O, more than might be expected.
An `alarm(2)` system call for 3600 seconds is made (and caught) within the `dial` module for the purpose of "touching" the `LCK..` file and constitutes the device allocation semaphore for the terminal device. Otherwise, `uucp(1C)` may simply delete the `LCK..` entry on its 90-minute clean-up rounds. The alarm may go off while the user program is in a `read(2)` or `write(2)` system call, causing an apparent error return. If the user program expects to be around for an hour or more, error returns from reads should be checked for (`errno==EINTR`), and the `read` possibly reissued.
NAME
drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48,
lcong48 — generate uniformly distributed pseudo-random numbers

SYNOPSIS
double drand48 ( )
double erand48 (xsubi)
unsigned short xsubi[3];
long lrand48 ( )
long nrand48 (xsubi)
unsigned short xsubi[3];
long mrand48 ( )
long jrand48 (xsubi)
unsigned short xsubi[3];
void srand48 (seedval)
long seedval;
unsigned short *seed48 (seed16v)
unsigned short seed16v[3];
void lcong48 (param)
unsigned short param[7];

DESCRIPTION
This family of functions generates pseudo-random numbers using the well-
known linear congruential algorithm and 48-bit integer arithmetic.
Functions drand48 and erand48 return non-negative double-precision floating-
point values uniformly distributed over the interval [0.0, 1.0).
Functions lrand48 and nrand48 return non-negative long integers uniformly dis-
tributed over the interval [0, 2^31).
Functions mrand48 and jrand48 return signed long integers uniformly distributed
over the interval [-2^31, 2^31).
Functions srand48, seed48 and lcong48 are initialization entry points, one of
which should be invoked before either drand48, lrand48 or mrand48 is called.
(Although it is not recommended practice, constant default initializer values will
be supplied automatically if drand48, lrand48 or mrand48 is called without a prior
call to an initialization entry point.) Functions erand48, nrand48 and jrand48 do
not require an initialization entry point to be called first.
All the routines work by generating a sequence of 48-bit integer values, X_i,
according to the linear congruential formula
X_{i+1} = (aX_i + c) mod m \quad n \geq 0.
The parameter m = 2^{48}; hence 48-bit integer arithmetic is performed. Unless
lcong48 has been invoked, the multiplier value a and the addend value c are
given by
a = \text{5DEECE66D}_{16} = 273673163155_{8}
\[ c = B_{16} = 13_8. \]

The value returned by any of the functions `drand48`, `erand48`, `lrand48`, `nrand48`, `mrand48` or `jrand48` is computed by first generating the next 48-bit \( X_i \) in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of \( X_i \) and transformed into the returned value.

The functions `drand48`, `lrand48` and `mrand48` store the last 48-bit \( X_i \) generated in an internal buffer, and must be initialized prior to being invoked. The functions `erand48`, `nrand48` and `jrand48` require the calling program to provide storage for the successive \( X_i \) values in the array specified as an argument when the functions are invoked. These routines do not have to be initialized; the calling program must place the desired initial value of \( X_i \) into the array and pass it as an argument. By using different arguments, functions `erand48`, `nrand48` and `jrand48` allow separate modules of a large program to generate several independent streams of pseudo-random numbers, i.e., the sequence of numbers in each stream will not depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function `srand48` sets the high-order 32 bits of \( X_i \) to the 32 bits contained in its argument. The low-order 16 bits of \( X_i \) are set to the arbitrary value \( 330E_{16} \).

The initializer function `seed48` sets the value of \( X_i \) to the 48-bit value specified in the argument array. In addition, the previous value of \( X_i \) is copied into a 48-bit internal buffer, used only by `seed48`, and a pointer to this buffer is the value returned by `seed48`. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time — use the pointer to get at and store the last \( X_i \) value, and then use this value to reinitialize via `seed48` when the program is restarted.

The initialization function `lcong48` allows the user to specify the initial \( X_i \), the multiplier value \( a \), and the addend value \( c \). Argument array elements `param[0-2]` specify \( X_i \), `param[3-5]` specify the multiplier \( a \), and `param[6]` specifies the 16-bit addend \( c \). After `lcong48` has been called, a subsequent call to either `srand48` or `seed48` will restore the "standard" multiplier and addend values, \( a \) and \( c \), specified on the previous page.

### NOTES

The source code for the portable version can be used on computers which do not have floating-point arithmetic. In such a situation, functions `drand48` and `erand48` are replaced by the two new functions below.

```c
long irand48 (m)
unsigned short m;

long krand48 (xsubi, m)
unsigned short xsubi[3], m;
```

Functions `irand48` and `krand48` return non-negative long integers uniformly distributed over the interval \([0, m-1]\).

### SEE ALSO

rand(3C).
NAME
dup2 — duplicate an open file descriptor

SYNOPSIS
   int dup2 (fildes, fildes2)
   int fildes, fildes2;

DESCRIPTION
   Fildes is a file descriptor referring to an open file, and fildes2 is a non-negative
   integer less than NOFILES. dup2 causes fildes2 to refer to the same file as fildes.
   If fildes2 already referred to an open file, it is closed first.

   dup2 will fail if one or more of the following are true:

   [EBADF] Fildes is not a valid open file descriptor.
   [EMFILE] NOFILES file descriptors are currently open.

SEE ALSO
   creat(2), close(2), exec(2), fcntl(2), open(2), pipe(2), lockf(3C).

DIAGNOSTICS
   Upon successful completion a non-negative integer, namely the file descriptor, is
   returned. Otherwise, a value of -1 is returned and errno is set to indicate the
   error.
NAME
ecvt, fcvt, gcvt — convert floating-point number to string

SYNOPSIS
char *ecvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;
char *fcvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;
char *gcvt (value, ndigit, buf)
double value;
int ndigit;
char *buf;

DESCRIPTION
ecvt converts value to a null-terminated string of ndigit digits and returns a
pointer thereto. The high-order digit is non-zero, unless the value is zero. The
low-order digit is rounded. The position of the decimal point relative to the
beginning of the string is stored indirectly through decpt (negative means to the
left of the returned digits). The decimal point is not included in the returned
string. If the sign of the result is negative, the word pointed to by sign is non-
zero, otherwise it is zero.

Fcvt is identical to ecvt, except that the correct digit has been rounded for printf
"%f" (FORTRAN F-format) output of the number of digits specified by ndigit.

Gcvt converts the value to a null-terminated string in the array pointed to by buf
and returns buf. It attempts to produce ndigit significant digits in FORTRAN F-
format if possible, otherwise E-format, ready for printing. A minus sign, if there
is one, or a decimal point will be included as part of the returned string.
Trailing zeros are suppressed.

SEE ALSO
printf(3S).

BUGS
The values returned by ecvt and fcvt point to a single static data array whose
content is overwritten by each call.
NAME
   end, etext, edata — last locations in program

SYNOPSIS
   extern end;
   extern etext;
   extern edata;

DESCRIPTION
   These names refer neither to routines nor to locations with interesting contents. The address of etext is the first address above the program text, edata above the initialized data region, and end above the uninitialized data region.

   When execution begins, the program break (the first location beyond the data) coincides with end, but the program break may be reset by the routines of brk(2), malloc(3C), standard input/output [stdio(3S)], the profile (-p) option of cc(1), and so on. Thus, the current value of the program break should be determined by sbrk (char *) (0) [see brk(2)].

SEE ALSO
   cc(1), brk(2), malloc(3C), stdio(3S).
NAME
fclose, fflush — close or flush a stream

SYNOPSIS
#include <stdio.h>
int fclose (stream)
FILE *stream;
int fflush (stream)
FILE *stream;

DESCRIPTION
fclose causes any buffered data for the named stream to be written out, and the stream to be closed.
fclose is performed automatically for all open files upon calling exit(2).
flush causes any buffered data for the named stream to be written to that file. The stream remains open.

SEE ALSO
close(2), exit(2), fopen(3S), setbuf(3S), stdio(3S).

DIAGNOSTICS
These functions return 0 for success, and EOF if any error (such as trying to write to a file that has not been opened for writing) was detected.
NAME
    ferror, feof, clearerr, fileno — stream status inquiries

SYNOPSIS
    #include <stdio.h>
    int ferror (stream)
       FILE *stream;
    int feof (stream)
       FILE *stream;
    void clearerr (stream)
       FILE *stream;
    int fileno (stream)
       FILE *stream;

DESCRIPTION
    ferror returns non-zero when an I/O error has previously occurred reading from
    or writing to the named stream, otherwise zero.
    Feof returns non-zero when EOF has previously been detected reading the
    named input stream, otherwise zero.
    Clearerr resets the error indicator and EOF indicator to zero on the named
    stream.
    Fileno returns the integer file descriptor associated with the named stream; see
    open(2).

NOTES
    All these functions are implemented as macros; they cannot be declared or rede-
   clared.

SEE ALSO
    open(2), fopen(3S), stdio(3S).
NAME
fopen, freopen, fdopen — open a stream

SYNOPSIS
#include <stdio.h>
FILE *fopen (filename, type)
char *filename, *type;
FILE *freopen (filename, type, stream)
char *filename, *type;
FILE *stream;
FILE *fdopen (fildes, type)
int fildes;
char *type;

DESCRIPTION
fopen opens the file named by filename and associates a stream with it. fopen
returns a pointer to the FILE structure associated with the stream.
Filename points to a character string that contains the name of the file to be
opened.
Type is a character string having one of the following values:
"r" open for reading
"w" truncate or create for writing
"a" append; open for writing at end of file, or create for writing
"r+" open for update (reading and writing)
"w+" truncate or create for update
"a+" append; open or create for update at end-of-file

Freopen substitutes the named file in place of the open stream. The original
stream is closed, regardless of whether the open ultimately succeeds. Freopen
returns a pointer to the FILE structure associated with stream.
Freopen is typically used to attach the preopened streams associated with stdin,
stdout and stderr to other files.

Fdopen associates a stream with a file descriptor. File descriptors are obtained
from open, dup, creat, or pipe(2), which open files but do not return pointers to a
FILE structure stream. Streams are necessary input for many of the Section 3S
library routines. The type of stream must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the
resulting stream. However, output may not be directly followed by input
without an intervening fseek or rewind, and input may not be directly followed
by output without an intervening fseek, rewind, or an input operation which
encounters end-of-file.

When a file is opened for append (i.e., when type is "a" or "a+"), it is impossible
to overwrite information already in the file. fseek may be used to reposition the
file pointer to any position in the file, but when output is written to the file, the
current file pointer is disregarded. All output is written at the end of the file and
causes the file pointer to be repositioned at the end of the output. If two separate processes open the same file for append, each process may write freely to the file without fear of destroying output being written by the other. The output from the two processes will be intermixed in the file in the order in which it is written.

SEE ALSO
creat(2), dup(2), open(2), pipe(2), fclose(3S), fseek(3S), stdio(3S).

DIAGNOSTICS
fopen, fdopen, and freopen return a NULL pointer on failure.
NAME
fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky — IEEE floating point environment control

SYNOPSIS
#include <ieeefp.h>
typedef enum {
    FP_RN=0,    /* round to nearest */
    FP_RP,      /* round to plus */
    FP_RM,      /* round to minus */
    FP_RZ,      /* round to zero (truncate) */
} fp_rnd;

fp_rnd fpgetround0;

fp_rnd fpsetround(rnd_dir)
fp_rnd rnd_dir;

#define fp_except int
#define FP_X_INV 0x10    /* invalid operation exception*/
#define FP_X_OFL 0x08    /* overflow exception*/
#define FP_X_UFL 0x04    /* underflow exception*/
#define FP_X_DZ  0x02    /* divide-by-zero exception*/
#define FP_X_IMP 0x01    /* imprecise (loss of precision)*/

fp_except fpgetmask0;

fp_except fpsetmask(mask);
fp_except mask;

fp_except fpgetsticky0;

fp_except fpsetsticky(sticky);
fp_except sticky;

DESCRIPTION
There are five floating point exceptions: divide-by-zero, overflow, underflow, imprecise (inexact) result, and invalid operation. When a floating point exception occurs, the corresponding sticky bit is set (1), and if the mask bit is enabled (1), the trap takes place. These routines let the user change the behavior on occurrence of any of these exceptions, as well as change the rounding mode for floating point operations.

fpgetround() returns the current rounding mode.

fpsetround() sets the rounding mode and returns the previous rounding mode.

fpgetmask() returns the current exception masks.

fpsetmask() sets the exception masks and returns the previous setting.
fpgetsticky() returns the current exception sticky flags.

fpsetsticky() sets (clears) the exception sticky flags and returns the previous setting.

The default environment on the 3B computer family is:

- Rounding mode set to nearest (FP_RN),
- Divide-by-zero,
- Floating point overflow, and
- Invalid operation traps enabled.

SEE ALSO
isnan(3C).

WARNINGS
fpsetsticky() modifies all sticky flags. fpsetmask() changes all mask bits.

Both C and F77 require truncation (round to zero) for floating point to integral conversions. The current rounding mode has no effect on these conversions.

CAVEATS
One must clear the sticky bit to recover from the trap and to proceed. If the sticky bit is not cleared before the next trap occurs, a wrong exception type may be signaled.

For the same reason, when calling fpsetmask() the user should make sure that the sticky bit corresponding to the exception being enabled is cleared.
NAME
fread, fwrite — binary input/output

SYNOPSIS
#include <stdio.h>
#include <sys/types.h>
int fread (ptr, size, nitems, stream)
char *ptr;
int nitems;
size_t size;
FILE *stream;
int fwrite (ptr, size, nitems, stream)
char *ptr;
int nitems;
size_t size;
FILE *stream;

DESCRIPTION
fread copies, into an array pointed to by ptr, nitems items of data from the
named input stream, where an item of data is a sequence of bytes (not neces­sarily terminated by a null byte) of length size. fread stops appending bytes if
an end-of-file or error condition is encountered while reading stream, or if nitems
items have been read. fread leaves the file pointer in stream, if defined, pointing
to the byte following the last byte read if there is one. fread does not change
the contents of stream.

fwrite appends at most nitems items of data from the array pointed to by ptr to
the named output stream. fwrite stops appending when it has appended nitems
items of data or if an error condition is encountered on stream. fwrite does not
change the contents of the array pointed to by ptr.

The argument size is typically sizeof(*ptr) where the pseudo-function sizeof
specifies the length of an item pointed to by ptr. If ptr points to a data type
other than char it should be cast into a pointer to char.

SEE ALSO
read(2), write(2), fopen(3S), getc(3S), gets(3S), printf(3S), putc(3S), puts(3S),
scanf(3S), stdio(3S).

DIAGNOSTICS
fread and fwrite return the number of items read or written. If nitems is non­positive, no characters are read or written and 0 is returned by both fread and
fwrite.
NAME
frexp, ldexp, modf — manipulate parts of floating-point numbers

SYNOPSIS

```c
double frexp (value, eptr)
double value;
int *eptr;
double ldexp (value, exp)
double value;
int exp;
double modf (value, iptr)
double value, *iptr;
```

DESCRIPTION
Every non-zero number can be written uniquely as \( x \times 2^n \), where the "mantissa" (fraction) \( x \) is in the range \( 0.5 < |x| < 1.0 \), and the "exponent" \( n \) is an integer. `frexp` returns the mantissa of a double `value`, and stores the exponent indirectly in the location pointed to by `eptr`. If `value` is zero, both results returned by `frexp` are zero.

`ldexp` returns the quantity `value \times 2^{\exp}`.

`modf` returns the signed fractional part of `value` and stores the integral part indirectly in the location pointed to by `iptr`.

DIAGNOSTICS
If `ldexp` would cause overflow, \( \pm \text{HUGE} \) (defined in `<math.h>`) is returned (according to the sign of `value`), and `errno` is set to `ERANGE`. If `ldexp` would cause underflow, zero is returned and `errno` is set to `ERANGE`. 
NAME
fseek, rewind, ftell — reposition a file pointer in a stream

SYNOPSIS
#include <stdio.h>

int fseek (stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

void rewind (stream)
FILE *stream;

long ftell (stream)
FILE *stream;

DESCRIPTION
fseek sets the position of the next input or output operation on the stream. The
new position is at the signed distance offset bytes from the beginning, from the
current position, or from the end of the file, according as ptrname has the value
0, 1, or 2.

Rewind(stream) is equivalent to fseek(stream, 0L, 0), except that no value is
returned.

fseek and rewind undo any effects of ungetc(3S).

After fseek or rewind, the next operation on a file opened for update may be
either input or output.

Ftell returns the offset of the current byte relative to the beginning of the file
associated with the named stream.

SEE ALSO
fseek(2), fopen(3S), popen(3S), stdio(3S), ungetc(3S).

DIAGNOSTICS
fseek returns non-zero for improper seeks, otherwise zero. An improper seek
can be, for example, an fseek done on a file that has not been opened via fopen;
in particular, fseek may not be used on a terminal, or on a file opened via
popen(3S).

WARNING
Although on the UNIX system an offset returned by ftell is measured in bytes,
and it is permissible to seek to positions relative to that offset, portability to
non-UNIX systems requires that an offset be used by fseek directly. Arithmetic
may not meaningfully be performed on such an offset, which is not necessarily
measured in bytes.
NAME

ftw — walk a file tree

SYNOPSIS

include <ftw.h>

int ftw (path, fn, depth)
    char *path;
    int (*fn) ( );
    int depth;

DESCRIPTION

ftw recursively descends the directory hierarchy rooted in path. For each object in the hierarchy, ftw calls fn, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a stat structure [see stat(2)] containing information about the object, and an integer. Possible values of the integer, defined in the <ftw.h> header file, are FTW_F for a file, FTW_D for a directory, FTW_DNR for a directory that cannot be read, and FTW_NS for an object for which stat could not successfully be executed. If the integer is FTW_DNR, descendants of that directory will not be processed. If the integer is FTW_NS, the stat structure will contain garbage. An example of an object that would cause FTW_NS to be passed to fn would be a file in a directory with read but without execute (search) permission.

ftw visits a directory before visiting any of its descendants.

The tree traversal continues until the tree is exhausted, an invocation of fn returns a nonzero value, or some error is detected within ftw (such as an I/O error). If the tree is exhausted, ftw returns zero. If fn returns a nonzero value, ftw stops its tree traversal and returns whatever value was returned by fn. If ftw detects an error, it returns -1, and sets the error type in errno.

ftw uses one file descriptor for each level in the tree. The depth argument limits the number of file descriptors so used. If depth is zero or negative, the effect is the same as if it were 1. Depth must not be greater than the number of file descriptors currently available for use. ftw will run more quickly if depth is at least as large as the number of levels in the tree.

SEE ALSO

stat(2), malloc(3C).

BUGS

Because ftw is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

CAVEAT

ftw uses malloc(3C) to allocate dynamic storage during its operation. If ftw is forcibly terminated, such as by longjmp being executed by fn or an interrupt routine, ftw will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have fn return a nonzero value at its next invocation.
NAME
getc, getchar, fgetc, getw — get character or word from a stream

SYNOPSIS
#include <stdio.h>
int getc (stream)
FILE *stream;
int getchar ()
int fgetc (stream)
FILE *stream;
int getw (stream)
FILE *stream;

DESCRIPTION
getc returns the next character (i.e., byte) from the named input stream, as an
integer. It also moves the file pointer, if defined, ahead one character in stream.
getchar is defined as getc(stdin). getc and getchar are macros.

fgetc behaves like getc, but is a function rather than a macro. fgetc runs more
slowly than getc, but it takes less space per invocation and its name can be
passed as an argument to a function.

getw returns the next word (i.e., integer) from the named input stream. Getw
increments the associated file pointer, if defined, to point to the next word. The
size of a word is the size of an integer and varies from machine to machine.
Getw assumes no special alignment in the file.

SEE ALSO
fclose(3S), ferror(3S), fopen(3S), fread(3S), gets(3S), putc(3S), scanf(3S),
stdio(3S).

DIAGNOSTICS
These functions return the constant EOF at end-of-file or upon an error. Because
EOF is a valid integer, ferror(3S) should be used to detect getw errors.

WARNING
If the integer value returned by getc, getchar, or fgetc is stored into a character
variable and then compared against the integer constant EOF, the comparison
may never succeed, because sign-extension of a character on widening to integer
is machine-dependent.

CAVEATS
Because it is implemented as a macro, getc evaluates a stream argument more
than once. In particular, getc(*f++) does not work sensibly. fgetc should be
used instead.

Because of possible differences in word length and byte ordering, files written
using putw are machine-dependent, and may not be read using getw on a
different processor.
NAME
getcwd — get path-name of current working directory

SYNOPSIS
char *getcwd (buf, size)
char *buf;
int size;

DESCRIPTION
getcwd returns a pointer to the current directory path name. The value of size
must be at least two greater than the length of the path-name to be returned.
If buf is a NULL pointer, getcwd will obtain size bytes of space using malloc(3C).
In this case, the pointer returned by getcwd may be used as the argument in a
subsequent call to free.
The function is implemented by using popen(3S) to pipe the output of the
pwd(1) command into the specified string space.

EXAMPLE

void exit(), perror();
.
.
if ((cwd = getcwd((char *)NULL, 64)) == NULL) {
    perror("pwd");
    exit(2);
}
printf("%s\n", cwd);

SEE ALSO
malloc(3C), popen(3S).

DIAGNOSTICS
Returns NULL with errno set if size is not large enough, or if an error occurs in a
lower-level function.
NAME
getenv – return value for environment name

SYNOPSIS
char *getenv (name)
char *name;

DESCRIPTION
getenv searches the environment list [see environ(5)] for a string of the form
name=value, and returns a pointer to the value in the current environment if
such a string is present, otherwise a NULL pointer.

SEE ALSO
exec(2), putenv(3C), environ(5).
GETGREN T(3C) (C Programming Language Utilities)

NAME
getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent — get group file entry

SYNOPSIS
#include <grp.h>

struct group *getgrent ( )
struct group *getgrgid (gid)
    int gid;
struct group *getgrnam (name)
    char *name;
void setgrent ( )
void endgrent ( )
struct group *fgetgrent (f)
FILE *f;

DESCRIPTION
getgrent, getgrgid and getgrnam each return pointers to an object with the following structure containing the broken-out fields of a line in the /etc/group file. Each line contains a "group" structure, defined in the <grp.h> header file.

    struct group {
        char *gr_name;    /* the name of the group */
        char *gr_passwd;  /* the encrypted group password */
        int   gr_gid;     /* the numerical group ID */
        char **gr_mem;    /* vector of pointers to member names */
    };

getgrent when first called returns a pointer to the first group structure in the file; thereafter, it returns a pointer to the next group structure in the file; so, successive calls may be used to search the entire file. Getgrgid searches from the beginning of the file until a numerical group id matching gid is found and returns a pointer to the particular structure in which it was found. Getgrnam searches from the beginning of the file until a group name matching name is found and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to setgrent has the effect of rewinding the group file to allow repeated searches. Endgrent may be called to close the group file when processing is complete.

fgetgrent returns a pointer to the next group structure in the stream f, which matches the format of /etc/group.

FILES
/etc/group
SEE ALSO
getlogin(3C), getpwent(3C), group(4).

DIAGNOSTICS
A NULL pointer is returned on EOF or error.

WARNING
The above routines use <stdio.h>, which causes them to increase the size of
programs, not otherwise using standard I/O, more than might be expected.

CAVEAT
All information is contained in a static area, so it must be copied if it is to be
saved.
NAME
getlogin — get login name

SYNOPSIS
char *getlogin ( );

DESCRIPTION
getlogin returns a pointer to the login name as found in /etc/utmp. It may be
used in conjunction with getpwnam to locate the correct password file entry
when the same user ID is shared by several login names.

If getlogin is called within a process that is not attached to a terminal, it returns
a NULL pointer. The correct procedure for determining the login name is to call
cuserid, or to call getlogin and if it fails to call getpwuid.

FILES
/etc/utmp

SEE ALSO
cuserid(3S), getgrent(3C), getpwent(3C), utmp(4).

DIAGNOSTICS
Returns the NULL pointer if name is not found.

CAVEAT
The return values point to static data whose content is overwritten by each call.
NAME
getopt — get option letter from argument vector

SYNOPSIS
int getopt (argc, argv, optstring)

int argc;
char **argv, *opstring;
extern char *optarg;
extern int optind, opterr;

DESCRIPTION
getopt returns the next option letter in argv that matches a letter in optstring. It supports all the rules of the command syntax standard (see intro(1)). So all new commands will adhere to the command syntax standard, they should use getopts(1) or getopt(3C) to parse positional parameters and check for options that are legal for that command.

optstring must contain the option letters the command using getopt will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

optarg is set to point to the start of the option-argument on return from getopt.

getopt places in optind the argv index of the next argument to be processed.

optind is external and is initialized to 1 before the first call to getopt.

When all options have been processed (i.e., up to the first non-option argument), getopt returns -1. The special option "--" may be used to delimit the end of the options; when it is encountered, -1 will be returned, and "--" will be skipped.

DIAGNOSTICS
getopt prints an error message on standard error and returns a question mark (?) when it encounters an option letter not included in optstring or no option-argument after an option that expects one. This error message may be disabled by setting opterr to 0.

EXAMPLE
The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options a and b, and the option o, which requires an option-argument:

main (argc, argv)
int argc;
char **argv;
{
    int c;
    extern char *optarg;
    extern int optind;
    .

while ((c = getopt(argc, argv, "abo:")) != -1) {
  switch (c) {
    case 'a':
      if (bflg)
        errflg++;
      else
        aflg++;
      break;
    case 'b':
      if (aflg)
        errflg++;
      else
        bproc( );
      break;
    case 'o':
      ofile = optarg;
      break;
    case '?':
      errflg++;
  }
  if (errflg) {
    (void)fprintf(stderr, "usage: . . . ");
    exit (2);
  }
  for (; optind < argc; optind++) {
    ...
  }
}

This code will accept any of the following as equivalent:

```
cmd -a -b -o "xxx z yy" file
```

```
cmd -a -b -o "xxx z yy" -- file
```

```
cmd -ab -o xxx,z,yy file
```

```
cmd -ab -o "xxx z yy" file
```

```
cmd -o xxx,z,yy -b -a file
```

**WARNING**

Although the following command syntax rule (see `intro(1)`) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the EXAMPLE section above, a and b are options, and the option o requires an option-argument:

```
cmd -a oxxx file   (Rule 5 violation: options with option-arguments must not be grouped with other options)
```

```
cmd -ab -oxxx file   (Rule 6 violation: there must be white space after an option that takes an option-argument)
```

Changing the value of the variable `optind`, or calling `getopt` with different values of `argv`, may lead to unexpected results.

**SEE ALSO**

NAME
getpass — read a password

SYNOPSIS
char *getpass (prompt)
char *prompt;

DESCRIPTION
getpass reads up to a newline or EOF from the file /dev/tty, after prompting on
the standard error output with the null-terminated string prompt and disabling
echoing. A pointer is returned to a null-terminated string of at most 8 charac-
ters. If /dev/tty cannot be opened, a NULL pointer is returned. An interrupt
will terminate input and send an interrupt signal to the calling program before
returning.

FILES
/dev/tty

WARNING
The above routine uses <stdio.h>, which causes it to increase the size of pro-
grams not otherwise using standard I/O, more than might be expected.

CAVEAT
The return value points to static data whose content is overwritten by each call.
NAME
getpw — get name from UID

SYNOPSIS
int getpw (uid, buf)
int uid;
char *buf;

DESCRIPTION
getpw searches the password file for a user id number that equals uid, copies the
line of the password file in which uid was found into the array pointed to by
buf, and returns 0. getpw returns non-zero if uid cannot be found.

This routine is included only for compatibility with prior systems and should not
be used; see getpwent(3C) for routines to use instead.

FILES
/etc/passwd

SEE ALSO
getpwent(3C), passwd(4).

DIAGNOSTICS
getpw returns non-zero on error.

WARNING
The above routine uses <stdio.h>, which causes it to increase, more than
might be expected, the size of programs not otherwise using standard I/O.
NAME
getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent — get password file entry

SYNOPSIS
#include <pwd.h>
struct passwd *getpwent ( )
struct passwd *getpwuid (uid)
    int uid;
struct passwd *getpwnam (name)
    char *name;
void setpwent ( )
void endpwent ( )
struct passwd *fgetpwent (f)
FILE *f;

DESCRIPTION
getpwent, getpwuid and getpwnam each returns a pointer to an object with the following structure containing the broken-out fields of a line in the /etc/passwd file. Each line in the file contains a "passwd" structure, declared in the <pwd.h> header file:
struct passwd {
    char   *pw_name;
    char   *pw_password;
    int    pw_uid;
    int    pw_gid;
    char   *pw_age;
    char   *pw_comment;
    char   *pw_gecos;
    char   *pw_dir;
    char   *pw_shell;
};

This structure is declared in <pwd.h> so it is not necessary to redefine it.
The fields have meanings described in passwd(4).
getpwent when first called returns a pointer to the first passwd structure in the file; thereafter, it returns a pointer to the next passwd structure in the file; so successive calls can be used to search the entire file. Getpwuid searches from the beginning of the file until a numerical user id matching uid is found and returns a pointer to the particular structure in which it was found. Getpwnam searches from the beginning of the file until a login name matching name is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.
A call to setpwent has the effect of rewinding the password file to allow repeated searches. Endpwent may be called to close the password file when processing is complete.
Fgetpwent returns a pointer to the next passwd structure in the stream \textit{f}, which matches the format of \texttt{/etc/passwd}.

\textbf{FILES}

\texttt{/etc/passwd}

\textbf{SEE ALSO}

getlogin(3C), getgrent(3C), passwd(4).

\textbf{DIAGNOSTICS}

A \texttt{NULL} pointer is returned on EOF or error.

\textbf{WARNING}

The above routines use \texttt{<stdio.h>}, which causes them to increase the size of programs, not otherwise using standard I/O, more than might be expected.

\textbf{CAVEAT}

All information is contained in a static area, so it must be copied if it is to be saved.
NAME
gets, fgets — get a string from a stream

SYNOPSIS
#include <stdio.h>
char *gets (s)
char *fgets (s, n, stream)
char *s;
int n;
FILE *stream;

DESCRIPTION
gets reads characters from the standard input stream, stdin, into the array pointed to by s, until a new-line character is read or an end-of-file condition is encountered. The new-line character is discarded and the string is terminated with a null character.

Fgets reads characters from the stream into the array pointed to by s, until n−1 characters are read, or a new-line character is read and transferred to s, or an end-of-file condition is encountered. The string is then terminated with a null character.

SEE ALSO
ferror(3S), fopen(3S), fread(3S), getc(3S), scanf(3S), stdio(3S).

DIAGNOSTICS
If end-of-file is encountered and no characters have been read, no characters are transferred to s and a NULL pointer is returned. If a read error occurs, such as trying to use these functions on a file that has not been opened for reading, a NULL pointer is returned. Otherwise s is returned.
NAME

getut: getutent, getutid, getutline, pututline, setutent, endutent, utmpname —
access utmp file entry

SYNOPSIS

#include <utmp.h>

struct utmp *getutent ( )
struct utmp *getutid (id)
struct utmp *id;
struct utmp *getutline (line)
struct utmp *line;
void pututline (utmp)
struct utmp *utmp;
void setutent ( )
void endutent ( )
void utmpname (file)
char *file;

DESCRIPTION

getutent, getutid and getutline each return a pointer to a structure of the fol­
lowing type:

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, lnxn) */
    short ut_pid; /* process id */
    short ut_type; /* type of entry */
    struct exit_status {
        short e_termination; /* Process termination status */
        short e_exit; /* Process exit status */
    } ut_exit; /* The exit status of a process */
    time_t ut_time; /* time entry was made */
};

getutent reads in the next entry from a utmp-like file. If the file is not already
open, it opens it. If it reaches the end of the file, it fails.

getutid searches forward from the current point in the utmp file until it finds an
entry with a ut_type matching id->ut_type if the type specified is RUN_LVL,
BOOT_TIME, OLD_TIME or NEW_TIME. If the type specified in id is
INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS or DEAD_PROCESS, then getutid
will return a pointer to the first entry whose type is one of these four and whose
ut_id field matches id->ut_id. If the end of file is reached without a match, it
fails.
getutline searches forward from the current point in the utmp file until it finds an entry of the type LOGIN_PROCESS or USER_PROCESS which also has a ut_line string matching the line->ut_line string. If the end of file is reached without a match, it fails.

Pututline writes out the supplied utmp structure into the utmp file. It uses getutid to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of pututline will have searched for the proper entry using one of the getut routines. If so, pututline will not search. If pututline does not find a matching slot for the new entry, it will add a new entry to the end of the file.

Setutent resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.

Endutent closes the currently open file.

Utmpname allows the user to change the name of the file examined, from /etc/utmp to any other file. It is most often expected that this other file will be /etc/wtmp. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. Utmpname does not open the file. It just closes the old file if it is currently open and saves the new file name.
NAME
hsearch, hcreate, hdestroy — manage hash search tables

SYNOPSIS
#include <search.h>
ENTRY *hsearch (item, action)
ENTRY item;
ACTION action;
int hcreate (nel)
unsigned nel;
void hdestroy ( )

DESCRIPTION
hsearch is a hash-table search routine generalized from Knuth (6.4) Algorithm D. It returns a pointer into a hash table indicating the location at which an entry can be found. Item is a structure of type ENTRY (defined in the <search.h> header file) containing two pointers: item.key points to the comparison key, and item.data points to any other data to be associated with that key. (Pointers to types other than character should be cast to pointer-to-character.) Action is a member of an enumeration type ACTION indicating the disposition of the entry if it cannot be found in the table. ENTER indicates that the item should be inserted in the table at an appropriate point. FIND indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a NULL pointer.

Hcreate allocates sufficient space for the table, and must be called before hsearch is used. Nel is an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

Hdestroy destroys the search table, and may be followed by another call to hcreate.

NOTES
hsearch uses open addressing with a multiplicative hash function. However, its source code has many other options available which the user may select by compiling the hsearch source with the following symbols defined to the preprocessor:

DIV Use the remainder modulo table size as the hash function instead of the multiplicative algorithm.
USCR Use a User Supplied Comparison Routine for ascertaining table membership. The routine should be named hcompar and should behave in a manner similar to strcmp [see string(3C)].
CHAINED Use a linked list to resolve collisions. If this option is selected, the following other options become available.
START Place new entries at the beginning of the linked list (default is at the end).
HSEARCH(3C)  (C Programming Language Utilities)  HSEARCH(3C)

**SORTUP**  Keep the linked list sorted by key in ascending order.

**SORTDOWN**  Keep the linked list sorted by key in descending order.

Additionally, there are preprocessor flags for obtaining debugging printout (−DDEBUG) and for including a test driver in the calling routine (−DDRIVER). The source code should be consulted for further details.

**EXAMPLE**

The following example will read in strings followed by two numbers and store them in a hash table, discarding duplicates. It will then read in strings and find the matching entry in the hash table and print it out.

```c
#include <stdio.h>
#include <search.h>

struct info { /* this is the info stored in the table */
    int age, room; /* other than the key. */
};

#define NUM_EMPL 5000 /* # of elements in search table */

main( )
{
    /* space to store strings */
    char string_space[NUM_EMPL*20];
    /* space to store employee info */
    struct info info_space[NUM_EMPL];
    /* next avail space in string_space */
    char *str_ptr = string_space;
    /* next avail space in info_space */
    struct info *info_ptr = info_space;
    ENTRY item, *found_item, *hsearch( );
    /* name to look for in table */
    char name_to_find[30];
    int i = 0;

    /* create table */
    (void) hcreate(NUM_EMPL);
    while (scanf("%s%d%d", str_ptr, &info_ptr->age,
               &info_ptr->room) != EOF & & i++ < NUM_EMPL) {
        /* put info in structure, and structure in item */
        item.key = str_ptr;
        item.data = (char *)info_ptr;
        str_ptr += strlen(str_ptr) + 1;
        info_ptr++;
        /* put item into table */
        (void) hsearch(item, ENTER);
    }

    /* access table */
```
item.key = name_to_find;
while (scanf("%s", item.key) != EOF) {
    if ((found_item = hsearch(item, FIND)) != NULL) {
        /* if item is in the table */
        (void)printf("found %s, age = %d, room = %d\n",
            found_item->key,
            ((struct info *)found_item->data)->age,
            ((struct info *)found_item->data)->room);
    } else {
        (void)printf("no such employee %s\n",
            name_to_find)
    }
}

SEE ALSO
bsearch(3C), lsearch(3C), malloc(3C), malloc(3X), string(3C), tsearch(3C).

DIAGNOSTICS
hsearch returns a NULL pointer if either the action is FIND and the item could
not be found or the action is ENTER and the table is full.
Hcreate returns zero if it cannot allocate sufficient space for the table.

WARNING
hsearch and hcreate use malloc(3C) to allocate space.

CAVEAT
Only one hash search table may be active at any given time.
NAME
isnan: isnand, isnanf — test for floating point NaN (Not-A-Number)

SYNOPSIS
#include <ieeefp.h>
int isnand (dsrc)
double dsrc;
int isnanf (fsrc)
float fsrc;

DESCRIPTION
isnan and isnanf return true (1) if the argument dsrc or fsrc is a NaN; otherwise
they return false (0).
Neither routine generates any exception, even for signaling NaNs.
isnanf() is implemented as a macro included in <ieeefp.h>.

SEE ALSO
fpgetround(3C).
NAME
l3tol, ltol3 — convert between 3-byte integers and long integers

SYNOPSIS
void l3tol (lp, cp, n)
long *lp;
char *cp;
int n;

void ltol3 (cp, lp, n)
char *cp;
long *lp;
int n;

DESCRIPTION
l3tol converts a list of \( n \) three-byte integers packed into a character string pointed to by \( cp \) into a list of long integers pointed to by \( lp \).

ltol3 performs the reverse conversion from long integers (\( lp \)) to three-byte integers (\( cp \)).

These functions are useful for file-system maintenance where the block numbers are three bytes long.

SEE ALSO
fs(4).

CAVEAT
Because of possible differences in byte ordering, the numerical values of the long integers are machine-dependent.
NAME
lockf — record locking on files

SYNOPSIS
#include <unistd.h>

int lockf (fildes, function, size)
long size;
int fildes, function;

DESCRIPTION
The lockf command will allow sections of a file to be locked; advisory or mandatory write locks depending on the mode bits of the file [see chmod(2)]. Locking calls from other processes which attempt to lock the locked file section will either return an error value or be put to sleep until the resource becomes unlocked. All the locks for a process are removed when the process terminates. [See fcntl(2) for more information about record locking.]

Fildes is an open file descriptor. The file descriptor must have O_WRONLY or O_RDWR permission in order to establish lock with this function call.

Function is a control value which specifies the action to be taken. The permissible values for function are defined in <unistd.h> as follows:

#define F _ULOCK 0 /* Unlock a previously locked section */
#define FLOCK 1 /* Lock a section for exclusive use */
#define FTLOCK 2 /* Test and lock a section for exclusive use */
#define F_TEST 3 /* Test section for other processes locks */

All other values of function are reserved for future extensions and will result in an error return if not implemented.

F_TEST is used to detect if a lock by another process is present on the specified section. F_LOCK and F_TLOCK both lock a section of a file if the section is available. F_ULOCK removes locks from a section of the file.

Size is the number of contiguous bytes to be locked or unlocked. The resource to be locked starts at the current offset in the file and extends forward for a positive size and backward for a negative size (the preceding bytes up to but not including the current offset). If size is zero, the section from the current offset through the largest file offset is locked (i.e., from the current offset through the present or any future end-of-file). An area need not be allocated to the file in order to be locked as such locks may exist past the end-of-file.

The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent sections occur, the sections are combined into a single section. If the request requires that a new element be added to the table of active locks and this table is already full, an error is returned, and the new section is not locked.

F_LOCK and F_TLOCK requests differ only by the action taken if the resource is not available. F_LOCK will cause the calling process to sleep until the resource is available. F_TLOCK will cause the function to return a −1 and set errno to [EACCES] error if the section is already locked by another process.
F_ULOCK requests may, in whole or in part, release one or more locked sections controlled by the process. When sections are not fully released, the remaining sections are still locked by the process. Releasing the center section of a locked section requires an additional element in the table of active locks. If this table is full, an [EDEADLK] error is returned and the requested section is not released.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by accessing another process's locked resource. Thus calls to lockf or fcnt! scan for a deadlock prior to sleeping on a locked resource. An error return is made if sleeping on the locked resource would cause a deadlock.

Sleeping on a resource is interrupted with any signal. The alarm(2) command may be used to provide a timeout facility in applications which require this facility.

The lockf utility will fail if one or more of the following are true:

[EBADF]  
  Fildes is not a valid open descriptor.

[EACCES]  
  Cmd is F_TLOCK or F_TEST and the section is already locked by another process.

[EDEADLK]  
  Cmd is F_LOCK and a deadlock would occur. Also the cmd is either F_LOCK, F_TLOCK, or F_ULOCK and the number of entries in the lock table would exceed the number allocated on the system.

[ECOMM]  
  Fildes is on a remote machine and the link to that machine is no longer active.

SEE ALSO
  chmod(2), close(2), creat(2), fcntl(2), intro(2), open(2), read(2), write(2).

DIAGNOSTICS
  Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

WARNINGS
  Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The standard I/O package is the most common source of unexpected buffering.

Because in the future the variable errno will be set to EAGAIN rather than EACCES when a section of a file is already locked by another process, portable application programs should expect and test for either value.

NAME
  lsearch, lfind — linear search and update

SYNOPSIS
  #include <stdio.h>
  #include <search.h>
char *lsearch ((char *)key, (char *)base, nelp, sizeof(*key), compar)
unsigned *nelp;
int (*compar)( );

char *lfind ((char *)key, (char *)base, nelp, sizeof(*key), compar)
unsigned *nelp;
int (*compar)( );

DESCRIPTION

lsearch is a linear search routine generalized from Knuth (6.1) Algorithm S. It returns a pointer into a table indicating where a datum may be found. If the datum does not occur, it is added at the end of the table. Key points to the datum to be sought in the table. Base points to the first element in the table. Nelp points to an integer containing the current number of elements in the table. The integer is incremented if the datum is added to the table. Compar is the name of the comparison function which the user must supply (strcmp, for example). It is called with two arguments that point to the elements being compared. The function must return zero if the elements are equal and non-zero otherwise.

lfind is the same as lsearch except that if the datum is not found, it is not added to the table. Instead, a NULL pointer is returned.

NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared. Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

EXAMPLE

This fragment will read in less than TABSIZE strings of length less than ELSIZE and store them in a table, eliminating duplicates.

```c
#include <stdio.h>
#include <search.h>

#define TABSIZE 50
#define ELSIZE 120

char line[ELSIZE], tab[TABSIZE][ELSIZE], *lsearch( );
unsigned nelp = 0;
int strcmp( );
```
while (fgets(line, ELSIZE, stdin) != NULL &&
       nel < TABSIZE)
    (void) lsearch(line, (char *)tab, &nel,
                   ELSIZE, strcmp);

SEE ALSO
bsearch(3C), hsearch(3C), string(3C), tsearch(3C).

DIAGNOSTICS
If the searched for datum is found, both lsearch and lfind return a pointer to it. Otherwise, lfind returns NULL and lsearch returns a pointer to the newly added element.

BUGS
Undefined results can occur if there is not enough room in the table to add a new item.
NAME
malloc, free, realloc, calloc — main memory allocator

SYNOPSIS
char *malloc (size)
unsigned size;
void free (ptr)
char *ptr;
char *realloc (ptr, size)
char *ptr;
unsigned size;
char *calloc (nelem, elsize)
unsigned nelem, elsize;

DESCRIPTION
malloc and free provide a simple general-purpose memory allocation package. malloc returns a pointer to a block of at least size bytes suitably aligned for any use.

The argument to free is a pointer to a block previously allocated by malloc; after free is performed this space is made available for further allocation, but its contents are left undisturbed.

Undefined results will occur if the space assigned by malloc is overrun or if some random number is handed to free.

malloc allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls sbrk [see brk(2)] to get more memory from the system when there is no suitable space already free.

realloc changes the size of the block pointed to by ptr to size bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. If no free block of size bytes is available in the storage arena, then realloc will ask malloc to enlarge the arena by size bytes and will then move the data to the new space.

realloc also works if ptr points to a block freed since the last call of malloc, realloc, or calloc; thus sequences of free, malloc and realloc can exploit the search strategy of malloc to do storage compaction.

calloc allocates space for an array of nelem elements of size elsize. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

SEE ALSO
brk(2), malloc(3X).

DIAGNOSTICS
malloc, realloc and calloc return a NULL pointer if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. When this happens the block pointed to by ptr may be destroyed.
NOTES
Search time increases when many objects have been allocated; that is, if a program allocates but never frees, then each successive allocation takes longer. For an alternate, more flexible implementation, see malloc(3X).
NAME
memory: memccpy, memchr, memcmp, memcpy, memset – memory operations

SYNOPSIS
#include <memory.h>

char *memccpy (s1, s2, c, n)
char *s1, *s2;
int c, n;

cchar *memchr (s, c, n)
char *s;
int c, n;

int memcmp (s1, s2, n)
char *s1, *s2;
int n;

char *memcpy (s1, s2, n)
char *s1, *s2;
int n;

char *memset (s, c, n)
char *s;
int c, n;

DESCRIPTION
These functions operate as efficiently as possible on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

Memccpy copies characters from memory area s2 into s1, stopping after the first occurrence of character c has been copied, or after n characters have been copied, whichever comes first. It returns a pointer to the character after the copy of c in s1, or a NULL pointer if c was not found in the first n characters of s2.

Memchr returns a pointer to the first occurrence of character c in the first n characters of memory area s, or a NULL pointer if c does not occur.

memcmp compares its arguments, looking at the first n characters only, and returns an integer less than, equal to, or greater than 0, according as s1 is lexicographically less than, equal to, or greater than s2.

Memcpy copies n characters from memory area s2 to s1. It returns s1.

Memset sets the first n characters in memory area s to the value of character c. It returns s.

For user convenience, all these functions are declared in the optional <memory.h> header file.
CAVEATS

`Memcmp` is implemented by using the most natural character comparison on the machine. Thus the sign of the value returned when one of the characters has its high order bit set is not the same in all implementations and should not be relied upon.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.
NAME
mktemp — make a unique file name

SYNOPSIS
char *mktemp (template)
char *template;

DESCRIPTION
mktemp replaces the contents of the string pointed to by template by a unique file name, and returns the address of template. The string in template should look like a file name with six trailing Xs; mktemp will replace the Xs with a letter and the current process ID. The letter will be chosen so that the resulting name does not duplicate an existing file.

SEE ALSO
getpid(2), tmpfile(3S), tmpnam(3S).

DIAGNOSTIC
mktemp will assign to template the NULL string if it cannot create a unique name.

CAVEAT
If called more than 17,576 time in a single process, this function will start recycling previously used names.
NAME
monitor — prepare execution profile

SYNOPSIS
#include <mon.h>

void monitor (lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)( ), (*highpc)( );
WORD *buffer;
int bufsize, nfunc;

DESCRIPTION
An executable program created by cc -p automatically includes calls for monitor
with default parameters; monitor need not be called explicitly except to gain fine
control over profiling.

monitor is an interface to profil(2). Lowpc and highpc are the addresses of two
functions; buffer is the address of a (user supplied) array of bufsize WORDs
(defined in the <mon.h> header file). monitor arranges to record a histogram of
periodically sampled values of the program counter, and of counts of calls of
certain functions, in the buffer. The lowest address sampled is that of lowpc
and the highest is just below highpc. Lowpc may not equal 0 for this use of monitor.
At most nfunc call counts can be kept; only calls of functions compiled with the
profiling option -p of cc(1) are recorded.

For the results to be significant, especially where there are small, heavily used
routines, it is suggested that the buffer be no more than a few times smaller than
the range of locations sampled.

To profile the entire program, it is sufficient to use

extern etext;

... monitor ((int (*)())2, &etext, buf, bufsize, nfunc);

Etext lies just above all the program text; see end(3C).

To stop execution monitoring and write the results, use

monitor ((int (*)())0, 0, 0, 0, 0);

Prof(1) can then be used to examine the results.

The name of the file written by monitor is controlled by the environment vari­
able PROFDIR. If PROFDIR does not exist, “mon.out” is created in the current
directory. If PROFDIR exists but has no value, monitor does not do any profiling
and creates no output file. Otherwise, the value of PROFDIR is used as the name
of the directory in which to create the output file. If PROFDIR is dirname, then
the file written is “dirname/pid.mon.out” where pid is the program’s process id.
(When monitor is called automatically by compiling via cc -p, the file created is
“dirname/pid.progname” where progname is the name of the program.)

FILES
mon.out

SEE ALSO
cc(1), prof(1), profi(2), end(3C).
BUGS

The "dirname/pid.mon.out" form does not work; the "dirname/pid.progname" form (automatically called via cc -p) does work.
NAME
nlist — get entries from name list

SYNOPSIS
#include <nlist.h>

int nlist (filename, nl)
char *filename;
struct nlist *nl;

DESCRIPTION
nlist examines the name list in the executable file whose name is pointed to by
filename, and selectively extracts a list of values and puts them in the array of
nlist structures pointed to by nl. The name list nl consists of an array of structures containing names of variables, types and values. The list is terminated
with a null name; that is, a null string is in the name position of the structure.
Each variable name is looked up in the name list of the file. If the name is
found, the type and value of the name are inserted in the next two fields. The
type field will be set to 0 unless the file was compiled with the -g option. If the
name is not found, both entries are set to 0. See a.out(4) for a discussion of the
symbol table structure.

This function is useful for examining the system name list kept in the file /unix.
In this way programs can obtain system addresses that are up to date.

NOTES
The <nlist.h> header file is automatically included by <a.out.h> for compatability. However, if the only information needed from <a.out.h> is for use of
nlist, then including <a.out.h> is discouraged. If <a.out.h> is included, the
line "#undef n_name" may need to follow it.

SEE ALSO
a.out(4).

DIAGNOSTICS
All value entries are set to 0 if the file cannot be read or if it does not contain a
valid name list.

nlist returns -1 upon error; otherwise it returns 0.
NAME
  perror, errno, sys_errlist, sys_nerr — system error messages

SYNOPSIS
  void perror (s)
  char *s;
  extern int errno;
  extern char *sys_errlist[];
  extern int sys_nerr;

DESCRIPTION
  perror produces a message on the standard error output, describing the last error
  encountered during a call to a system or library function. The argument string s
  is printed first, then a colon and a blank, then the message and a new-line.
  (However, if s="" the colon is not printed.) To be of most use, the argument
  string should include the name of the program that incurred the error. The error
  number is taken from the external variable errno, which is set when errors occur
  but not cleared when non-erroneous calls are made.

  To simplify variant formatting of messages, the array of message strings
  sys_errlist is provided; errno can be used as an index into this table to get the
  message string without the new-line. Sys_nerr is the number of messages in the
  table; it should be checked because new error codes may be added to the system
  before they are added to the table.

SEE ALSO
  intro(2).
NAME
popen, pclose — initiate pipe to/from a process

SYNOPSIS
#include <stdio.h>
FILE *popen (command, type)
char *command, *type;
int pclose (stream)
FILE *stream;

DESCRIPTION
popen creates a pipe between the calling program and the command to be executed. The arguments to popen are pointers to null-terminated strings. Command consists of a shell command line. Type is an I/O mode, either r for reading or w for writing. The value returned is a stream pointer such that one can write to the standard input of the command, if the I/O mode is w, by writing to the file stream; and one can read from the standard output of the command, if the I/O mode is r, by reading from the file stream.

A stream opened by popen should be closed by pclose, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type r command may be used as an input filter and a type w as an output filter.

EXAMPLE
A typical call may be:

char *cmd = "ls *.c";
FILE *ptr;
if ((ptr = popen(cmd, "r")) != NULL)
    while (fgets(buf, n, ptr) != NULL)
        (void) printf("%s", buf);

This will print in stdout [see stdio (3S)] all the file names in the current directory that have a ".c" suffix.

SEE ALSO
pipe(2), wait(2), fclose(3S), fopen(3S), stdio(3S), system(3S).

DIAGNOSTICS
popen returns a NULL pointer if files or processes cannot be created.

Pclose returns −1 if stream is not associated with a "popened" command.

WARNING
If the original and "popened" processes concurrently read or write a common file, neither should use buffered I/O, because the buffering gets all mixed up. Problems with an output filter may be forestalled by careful buffer flushing, e.g. with fflush [see fclose(3S)].
NAME
printf, fprintf, sprintf — print formatted output

SYNOPSIS
#include <stdio.h>
int printf (format , arg ... )
char *format;
int fprintf (stream, format , arg ... )
FILE *stream;
char *format;
int sprintf (s, format [, arg ] ... )
char *s, *format;

DESCRIPTION
printf places output on the standard output stream stdout. Fprintf places output on the named output stream. Sprintf places "output," followed by the null character (\0), in consecutive bytes starting at *s; it is the user’s responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the \0 in the case of sprintf), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its args under control of the format. The format is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more args. The results are undefined if there are insufficient args for the format. If the format is exhausted while args remain, the excess args are simply ignored.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

Zero or more flags, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag ‘-‘, described below, has been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.

A precision that gives the minimum number of digits to appear for the d, i, o, u, x, or X conversions, the number of digits to appear after the decimal point for the e, E, and f conversions, the maximum number of significant digits for the g and G conversion, or the maximum number of characters to be printed from a string in s conversion. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

An optional 1 (ell) specifying that a following d, i, o, u, x, or X conversion character applies to a long integer arg. An 1 before any other conversion character is ignored.
A character that indicates the type of conversion to be applied.

A field width or precision or both may be indicated by an asterisk (*) instead of a digit string. In this case, an integer arg supplies the field width or precision. The arg that is actually converted is not fetched until the conversion letter is seen, so the args specifying field width or precision must appear before the arg (if any) to be converted. A negative field width argument is taken as a ‘-’ flag followed by a positive field width. If the precision argument is negative, it will be changed to zero.

The flag characters and their meanings are:
- The result of the conversion will be left-justified within the field.
+ The result of a signed conversion will always begin with a sign (+ or -).
blank If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.
# This flag specifies that the value is to be converted to an “alternate form.” For c, d, i, s, and u conversions, the flag has no effect. For o conversion, it increases the precision to force the first digit of the result to be a zero. For x or X conversion, a non-zero result will have 0x or 0X prefixed to it. For e, E, f, g, and G conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For g and G conversions, trailing zeroes will not be removed from the result (which they normally are).

The conversion characters and their meanings are:

**d, i, o, u, x, X** The integer arg is converted to signed decimal (d or i), unsigned octal, (o), decimal (u), or hexadecimal notation (x or X), respectively; the letters abcdef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of zero is a null string.

**f** The float or double arg is converted to decimal notation in the style 
"[-]ddd.ddd," where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, six digits are output; if the precision is explicitly 0, no decimal point appears.

**e, E** The float or double arg is converted in the style "[-]d.ddde±dd," where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, six digits are produced; if the precision is zero, no decimal point appears. The E format code will produce a number with E instead of e introducing the exponent. The exponent always contains at least two digits.
PRINTF(3S)  
(C Programming Language Utilities)  
PRINTF(3S)

`g, G`
The float or double `arg` is printed in style `f` or `e` (or in style `E` in the case of a `G` format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style `e` will be used only if the exponent resulting from the conversion is less than \(-4\) or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

`c`
The character `arg` is printed.

`s`
The `arg` is taken to be a string (character pointer) and characters from the string are printed until a null character (`\0`) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A NULL value for `arg` will yield undefined results.

`%`
Print a `%`; no argument is converted.

In printing floating point types (float and double), if the exponent is `0x7FF` and the mantissa is not equal to zero, then the output is

\[-NaN0xddddddd\]

where `0xddddddd` is the hexadecimal representation of the leftmost 32 bits of the mantissa. If the mantissa is zero, the output is

\[
[±]inf.
\]

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by `printf` and `fprintf` are printed as if `putc(3S)` had been called.

EXAMPLES
To print a date and time in the form "Sunday, July 3, 10:02," where `weekday` and `month` are pointers to null-terminated strings:

```c
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);
```

To print \(\pi\) to 5 decimal places:

```c
printf("pi = %.5f", 4 * atan(1.0));
```

SEE ALSO
`ecvt(3C)`, `putc(3S)`, `scanf(3S)`, `stdio(3S)`.

322
NAME
putc, putchar, fputc, putw — put character or word on a stream

SYNOPSIS
#include <stdio.h>
int putc (c, stream)
int c;
FILE *stream;
int putchar (c)
int c;
int fputc (c, stream)
int c;
FILE *stream;
int putw (w, stream)
int w;
FILE *stream;

DESCRIPTION
putc writes the character c onto the output stream (at the position where the file
pointer, if defined, is pointing). putchar(c) is defined as putc(c, stdout). putc and
putchar are macros.

fputc behaves like putc, but is a function rather than a macro. fputc runs more
slowly than putc, but it takes less space per invocation and its name can be
passed as an argument to a function.

Putw writes the word (i.e. integer) w to the output stream (at the position at
which the file pointer, if defined, is pointing). The size of a word is the size of
an integer and varies from machine to machine. Putw neither assumes nor
causes special alignment in the file.

SEE ALSO
fclose(3S), ferror(3S), fopen(3S), fread(3S), printf(3S), puts(3S), setbuf(3S),
stdio(3S).

DIAGNOSTICS
On success, these functions (with the exception of putw) each return the value
they have written. [Putw returns ferror (stream)]. On failure, they return the con-
stant EOF. This will occur if the file stream is not open for writing or if the
output file cannot grow. Because EOF is a valid integer, ferror(3S) should be
used to detect putw errors.

CAVEATS
Because it is implemented as a macro, putc evaluates a stream argument more
than once. In particular, putc(c, *f++); doesn't work sensibly. fputc should be
used instead.
Because of possible differences in word length and byte ordering, files written
using putw are machine-dependent, and may not be read using getw on a
different processor.
NAME
putenv — change or add value to environment

SYNOPSIS
int putenv (string)
char *string;

DESCRIPTION
String points to a string of the form “name=value.” putenv makes the value of the environment variable name equal to value by altering an existing variable or creating a new one. In either case, the string pointed to by string becomes part of the environment, so altering the string will change the environment. The space used by string is no longer used once a new string-defining name is passed to putenv.

SEE ALSO
exec(2), getenv(3C), malloc(3C), environ(5).

DIAGNOSTICS
putenv returns non-zero if it was unable to obtain enough space via malloc for an expanded environment, otherwise zero.

WARNINGS
putenv manipulates the environment pointed to by environ, and can be used in conjunction with getenv. However, envp (the third argument to main) is not changed.
This routine uses malloc(3C) to enlarge the environment.
After putenv is called, environmental variables are not in alphabetical order.
A potential error is to call putenv with an automatic variable as the argument, then exit the calling function while string is still part of the environment.
NAME
putpwent — write password file entry

SYNOPSIS
#include <pwd.h>

int putpwent (p, f)
struct passwd *p;
FILE *f;

DESCRIPTION
putpwent is the inverse of getpwent(3C). Given a pointer to a passwd structure
created by getpwent (or getpwuid or getpwnam), putpwent writes a line on the
stream f, which matches the format of /etc/passwd.

SEE ALSO
getpwent(3C).

DIAGNOSTICS
putpwent returns non-zero if an error was detected during its operation, other­
wise zero.

WARNING
The above routine uses <stdio.h>, which causes it to increase the size of pro­
grams, not otherwise using standard I/O, more than might be expected.
NAME
   puts, fputs — put a string on a stream

SYNOPSIS
   #include <stdio.h>
   int puts (s)
   char *s;
   int fputs (s, stream)
   char *s;
   FILE *stream;

DESCRIPTION
   puts writes the null-terminated string pointed to by s, followed by a new-line
   character, to the standard output stream stdout.

   fputs writes the null-terminated string pointed to by s to the named output
   stream.

   Neither function writes the terminating null character.

SEE ALSO
   ferror(3S), fopen(3S), fread(3S), printf(3S), putc(3S), stdio(3S).

DIAGNOSTICS
   Both routines return EOF on error. This will happen if the routines try to write
   on a file that has not been opened for writing.

NOTES
   puts appends a new-line character while fputs does not.
NAME
qsort — quicker sort

SYNOPSIS
void qsort ((char *) base, nel, sizeof (*base), compar)
unsigned nel;
int (*compar)();

DESCRIPTION
qsort is an implementation of the quicker-sort algorithm. It sorts a table of data in place.

Base points to the element at the base of the table. Nel is the number of elements in the table. Compar is the name of the comparison function, which is called with two arguments that point to the elements being compared. As the function must return an integer less than, equal to, or greater than zero, so must the first argument to be considered be less than, equal to, or greater than the second.

NOTES
The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.
The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.
The order in the output of two items which compare as equal is unpredictable.

SEE ALSO
bsearch(3C), lsearch(3C), string(3C).
NAME
rand, srand — simple random-number generator

SYNOPSIS
int rand ( )
void srand (seed)
unsigned seed;

DESCRIPTION
rand uses a multiplicative congruential random-number generator with period
\(2^{32}\) that returns successive pseudo-random numbers in the range from 0 to
\(2^{15} - 1\).

Srand can be called at any time to reset the random-number generator to a
random starting point. The generator is initially seeded with a value of 1.

NOTES
The spectral properties of rand are limited. Drand48(3C) provides a much better,
though more elaborate, random-number generator.

SEE ALSO
drand48(3C).
NAME
    scanf, fscanf, sscanf — convert formatted input

SYNOPSIS
    #include <stdio.h>
    int scanf (format [ , pointer ] ... )
    char *format;
    int fscanf (stream, format [ , pointer ] ... )
    FILE *stream;
    char *format;
    int sscanf (s, format [ , pointer ] ... )
    char *s, *format;

DESCRIPTION
    scanf reads from the standard input stream stdin. fscanf reads from the named
    input stream. sscanf reads from the character string s. Each function reads char-
    acters, interprets them according to a format, and stores the results in its argu-
    ments. Each expects, as arguments, a control string format described below, and
    a set of pointer arguments indicating where the converted input should be
    stored. The results are undefined in there are insufficient args for the format. If
    the format is exhausted while args remain, the excess args are simply ignored.

    The control string usually contains conversion specifications, which are used to
direct interpretation of input sequences. The control string may contain:

    1. White-space characters (blanks, tabs, new-lines, or form-feeds) which, except
       in two cases described below, cause input to be read up to the next non-
       white-space character.
    2. An ordinary character (not %), which must match the next character of the
       input stream.
    3. Conversion specifications, consisting of the character %, an optional assign-
       ment suppressing character *, an optional numerical maximum field width, an
       optional l (ell) or h indicating the size of the receiving variable, and a
       conversion code.

    A conversion specification directs the conversion of the next input field; the
    result is placed in the variable pointed to by the corresponding argument, unless
    assignment suppression was indicated by *. The suppression of assignment pro-
    vides a way of describing an input field which is to be skipped. An input field is
defined as a string of non-space characters; it extends to the next inappropriate
    character or until the field width, if specified, is exhausted. For all descriptors
    except "[" and "c", white space leading an input field is ignored.

    The conversion code indicates the interpretation of the input field; the
    corresponding pointer argument must usually be of a restricted type. For a
    suppressed field, no pointer argument is given. The following conversion codes
    are legal:

    %       a single % is expected in the input at this point; no assignment is done.
    d       a decimal integer is expected; the corresponding argument should be an
            integer pointer.
u an unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.

o an octal integer is expected; the corresponding argument should be an integer pointer.

x a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

i an integer is expected; the corresponding argument should be an integer pointer. It will store the value of the next input item interpreted according to C conventions: a leading "0" implies octal; a leading "0x" implies hexadecimal; otherwise, decimal.

n stores in an integer argument the total number of characters (including white space) that have been scanned so far since the function call. No input is consumed.

e,f,g a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a float. The input format for floating point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of an E or an e, followed by an optional +, -, or space, followed by an integer.

s a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating \0, which will be added automatically. The input field is terminated by a white-space character.

c a character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next non-space character, use %1s. If a field width is given, the corresponding argument should refer to a character array; the indicated number of characters is read.

[ indicates string data and the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters, which we will call the scanset, and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the scanset. The circumflex (^), when it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters not contained in the remainder of the scanset string. There are some conventions used in the construction of the scanset. A range of characters may be represented by the construct first-last, thus [0123456789] may be expressed [0-9]. Using this convention, first must be lexically less than or equal to last, or else the dash will stand for itself. The dash will also stand for itself whenever it is the first or the last character in the scanset. To include the right square
bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanset, and in this case it will not be syntactically interpreted as the closing bracket. The corresponding argument must point to a character array large enough to hold the data field and the terminating \0, which will be added automatically. At least one character must match for this conversion to be considered successful.

The conversion characters d, u, o, x and i may be preceded by l or h to indicate that a pointer to long or to short rather than to int is in the argument list. Similarly, the conversion characters e, f, and g may be preceded by l to indicate that a pointer to double rather than to float is in the argument list. The l or h modifier is ignored for other conversion characters.

scanf conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

scanf returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, EOF is returned.

EXAMPLES
The call:

```c
int n ; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 thompson
```

will assign to n the value 3, to i the value 25, to x the value 5.432, and name will contain thompson\0 . Or:

```c
int i, j; float x; char name[50];
(void) scanf("%i%2d%f%*d %[0-9] ", &j, &i, &x, name);
```

with input:

```
011 56789 0123 56a72
```

will assign 9 to j, 56 to i, 789.0 to x, skip 0123, and place the string 56\0 in name. The next call to getchar [see getc(3S)] will return a. Or:

```c
int i, j, s, e; char name[50];
(void) scanf("%i %i %n%n%s%n", &i, &j, &s, name, &e);
```

with input:

```
0x11 0xy johnson
```

will assign 17 to i, 0 to j, 6 to s, will place the string xy\0 in name, and will assign 8 to e. Thus, the length of name is e - s = 2 . The next call to getchar [see getc(3S)] will return a blank.

SEE ALSO
getc(3S), printf(3S), stdio(3S), strtod(3C), strtol(3C).
DIAGNOSTICS
These functions return EOF on end of input and a short count for missing or illegal data items.

CAVEATS
Trailing white space (including a new-line) is left unread unless matched in the control string.
NAME

setbuf, setvbuf — assign buffering to a stream

SYNOPSIS

#include <stdio.h>

void setbuf (stream, buf)
FILE *stream;
char *buf;

int setvbuf (stream, buf, type, size)
FILE *stream;
char *buf;
int type, size;

DESCRIPTION

setbuf may be used after a stream has been opened but before it is read or written. It causes the array pointed to by buf to be used instead of an automatically allocated buffer. If buf is the NULL pointer input/output will be completely unbuffered.

A constant BUFSIZ, defined in the <stdio.h> header file, tells how big an array is needed:

    char buf[BUFSIZ];

Setvbuf may be used after a stream has been opened but before it is read or written. Type determines how stream will be buffered. Legal values for type (defined in stdio.h) are:

    _I_OFBF causes input/output to be fully buffered.
    _I_OLBF causes output to be line buffered; the buffer will be flushed when a newline is written, the buffer is full, or input is requested.
    _I_ONBF causes input/output to be completely unbuffered.

If buf is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. Size specifies the size of the buffer to be used. The constant BUFSIZ in <stdio.h> is suggested as a good buffer size. If input/output is unbuffered, buf and size are ignored.

By default, output to a terminal is line buffered and all other input/output is fully buffered.

SEE ALSO

fopen(3S), getc(3S), malloc(3C), putc(3S), stdio(3S).

DIAGNOSTICS

If an illegal value for type or size is provided, setvbuf returns a non-zero value. Otherwise, the value returned will be zero.

NOTES

A common source of error is allocating buffer space as an "automatic" variable in a code block, and then failing to close the stream in the same block.
NAME
setjmp, longjmp - non-local goto

SYNOPSIS
#include <setjmp.h>
int setjmp (env)
jmp_buf env;
void longjmp (env, val)
jmp_buf env;
int val;

DESCRIPTION
These functions are useful for dealing with errors and interrupts encountered in
a low-level subroutine of a program.

setjmp saves its stack environment in env (whose type, jmp_buf, is defined in the
<setjmp.h> header file) for later use by longjmp. It returns the value 0.

Longjmp restores the environment saved by the last call of setjmp with the
corresponding env argument. After longjmp is completed, program execution
continues as if the corresponding call of setjmp (which must not itself have
returned in the interim) had just returned the value val. Longjmp cannot cause
setjmp to return the value 0. If longjmp is invoked with a second argument of 0,
setjmp will return 1. At the time of the second return from setjmp, all accessible
data have values as of the time longjmp is called. However, global variables will
have the expected values, i.e. those as of the time of the longjmp (see example).

EXAMPLE
#include <setjmp.h>
jmp_buf env;
int i = 0;
main ()
{
    void exit();

    if(setjmp(env) != 0) {
        (void) printf("value of i on 2nd return from setjmp: %d\n", i);
        exit(0);
    }
    (void) printf("value of i on 1st return from setjmp: %d\n", i);
    i = 1;
g() :
    /*NOTREACHED*/
}

g()
{
    longjmp(env, 1);
    /*NOTREACHED*/
}
If the a.out resulting from this C language code is run, the output will be:

value of i on 1st return from setjmp: 0

value of i on 2nd return from setjmp: 1

SEE ALSO
signal(2).

WARNING
If `longjmp` is called even though `env` was never primed by a call to `setjmp`, or when the last such call was in a function which has since returned, absolute chaos is guaranteed.

BUGS
The values of the registers on the second return from `setjmp` are the register values at the time of the first call to `setjmp`, not those at the time of the `longjmp`. This means that variables in a given function may behave differently in the presence of `setjmp`, depending on whether they are register or stack variables.
NAME
sleep — suspend execution for interval

SYNOPSIS
unsigned sleep (seconds)
unsigned seconds;

DESCRIPTION
The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) Because scheduled wakeups occur at fixed 1-second intervals, (on the second, according to an internal clock) and (2) because any caught signal will terminate the sleep following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount due to the scheduling of other activity in the system. The value returned by sleep will be the “unslept” amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested sleep time, or premature arousal due to another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling sleep. If the sleep time exceeds the time till such alarm signal, the process sleeps only until the alarm signal would have occurred. The caller's alarm catch routine is executed just before the sleep routine returns. But if the sleep time is less than the time till such alarm, the prior alarm time is reset to go off at the same time it would have without the intervening sleep.

SEE ALSO
alarm(2), pause(2), signal(2).
NAME

ssignal, gsignal - software signals

SYNOPSIS

#include <signal.h>

int (*ssignal (sig, action))( )
int sig, (*action)();

int gsignal (sig)
int sig;

DESCRIPTION

ssignal and gsignal implement a software facility similar to signal(2). This facility is used by the Standard C Library to enable users to indicate the disposition of error conditions, and is also made available to users for their own purposes.

Software signals made available to users are associated with integers in the inclusive range 1 through 16. A call to ssignal associates a procedure, action, with the software signal sig; the software signal, sig, is raised by a call to gsignal. Raising a software signal causes the action established for that signal to be taken.

The first argument to ssignal is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a (user-defined) action function or one of the manifest constants SIG_DFL (default) or SIG_IGN (ignore). ssignal returns the action previously established for that signal type; if no action has been established or the signal number is illegal, ssignal returns SIG_DFL.

Gsignal raises the signal identified by its argument, sig:

If an action function has been established for sig, then that action is reset to SIG_DFL and the action function is entered with argument sig. Gsignal returns the value returned to it by the action function.

If the action for sig is SIG_IGN, gsignal returns the value 1 and takes no other action.

If the action for sig is SIG_DFL, gsignal returns the value 0 and takes no other action.

If sig has an illegal value or no action was ever specified for sig, gsignal returns the value 0 and takes no other action.

SEE ALSO

signal(2), sigset(2).

NOTES

There are some additional signals with numbers outside the range 1 through 16 which are used by the Standard C Library to indicate error conditions. Thus, some signal numbers outside the range 1 through 16 are legal, although their use may interfere with the operation of the Standard C Library.
NAME
stdio – standard buffered input/output package

SYNOPSIS
#include <stdio.h>
FILE *stdin, *stdout, *stderr;

DESCRIPTION
The functions described in the entries of sub-class 3S of this manual constitute
an efficient, user-level I/O buffering scheme. The in-line macros getc(3S) and
putc(3S) handle characters quickly. The macros getchar and putchar, and the
higher-level routines fgetc, fgets, fprintf, fputc, fputs, fread, fscanf, fwrite, gets,
getw, printf, puts, putw, and scanf all use or act as if they use getc and putc; they
can be freely intermixed.

A file with associated buffering is called a stream and is declared to be a pointer
to a defined type FILE. Fopen(3S) creates certain descriptive data for a stream
and returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the
<stdio.h> header file and associated with the standard open files:

    stdin   standard input file
    stdout  standard output file
    stderr  standard error file

A constant NULL (0) designates a nonexistent pointer.

An integer-constant EOF (-1) is returned upon end-of-file or error by most
integer functions that deal with streams (see the individual descriptions for
details).

An integer constant BUFSIZ specifies the size of the buffers used by the partic-
ular implementation.

Any program that uses this package must include the header file of pertinent
macro definitions, as follows:

    #include <stdio.h>

The functions and constants mentioned in the entries of sub-class 3S of this
manual are declared in that header file and need no further declaration. The
constants and the following "functions" are implemented as macros (redeclara-
tion of these names is perilous): getc, getchar, putc, putchar, ferror, feof, clearerr,
and fileno.

Output streams, with the exception of the standard error stream stderr, are by
default buffered if the output refers to a file and line-buffered if the output refers
to a terminal. The standard error output stream stderr is by default unbuffered,
but use of freopen [see fopen(3S)] will cause it to become buffered or line-
buffered. When an output stream is unbuffered, information is queued for
writing on the destination file or terminal as soon as written; when it is buffered,
many characters are saved up and written as a block. When it is
line-buffered, each line of output is queued for writing on the destination terminal as soon as the line is completed (that is, as soon as a new-line character is written or terminal input is requested). `setbuf(3S)` or `setvbuf()` in `setbuf(3S)` may be used to change the stream’s buffering strategy.

SEE ALSO
open(2), close(2), lseek(2), pipe(2), read(2), write(2), ctermid(3S), cuserid(3S), fclose(3S), ferror(3S), fopen(3S), fread(3S), fseek(3S), gets(3S), popen(3S), printf(3S), putc(3S), puts(3S), scanf(3S), setbuf(3S), system(3S), tmpfile(3S), tmpnam(3S), ungetc(3S).

DIAGNOSTICS
Invalid stream pointers will usually cause grave disorder, possibly including program termination. Individual function descriptions describe the possible error conditions.
NAME
stdipc: ftok — standard interprocess communication package

SYNOPSIS
#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(path, id)
char *path;
char id;

DESCRIPTION
All interprocess communication facilities require the user to supply a key to be
used by the msgget(2), semget(2), and shmget(2) system calls to obtain interpro­
cess communication identifiers. One suggested method for forming a key is to use
the ftok subroutine described below. Another way to compose keys is to include
the project ID in the most significant byte and to use the remaining portion as a sequence number. There are many other ways to form keys, but it is
necessary for each system to define standards for forming them. If some stan­
dard is not adhered to, it will be possible for unrelated processes to unintentionally interfere with each other’s operation. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys
do not conflict across a given system.

Ftok returns a key based on path and id that is usable in subsequent msgget,
semget, and shmget system calls. Path must be the path name of an existing file
that is accessible to the process. Id is a character which uniquely identifies a
project. Note that ftok will return the same key for linked files when called with
the same id and that it will return different keys when called with the same file
name but different ids.

SEE ALSO
intro(2), msgget(2), semget(2), shmget(2).

DIAGNOSTICS
Ftok returns (key_t) -1 if path does not exist or if it is not accessible to the pro­
cess.

WARNING
If the file whose path is passed to ftok is removed when keys still refer to the
file, future calls to ftok with the same path and id will return an error. If the
same file is recreated, then ftok is likely to return a different key than it did the
original time it was called.
NAME
string: strcat, strdup, strncat, strncmp, strcmp, strcpyp, strncmp, strcpy, strncpy, strlen, strchr,
strrchr, strpbrk, strspn, strcspn, strtok - string operations

SYNOPSIS
#include <string.h>
#include <sys/types.h>
char *strcat (s1, s2)
char *s1, *s2;
char *strdup (s1)
char *s1;
char *strncat (s1, s2, n)
char *s1, *s2;
size_t n;
int strcmp (s1, s2)
char *s1, *s2;
int strncmp (s1, s2, n)
char *s1, *s2;
size_t n;
char *strncpy (s1, s2)
char *s1, *s2;
char *strncpy (s1, s2, n)
char *s1, *s2;
size_t n;
int strlen (s)
char *s;
char *strchr (s, c)
char *s;
int c;
char *strrchr (s, c)
char *s;
int c;
char *strpbrk (s1, s2)
char *s1, *s2;
int strspn (s1, s2)
char *s1, *s2;
int strcspn (s1, s2)
char *s1, *s2;
char *strtok (s1, s2)
char *s1, *s2;

DESCRIPTION
The arguments s1, s2 and s point to strings (arrays of characters terminated by a
null character). The functions strcat, strncat, strcpyp, and strncpy all alter s1. These functions do not check for overflow of the array pointed to by s1.
**Strcat** appends a copy of string $s_2$ to the end of string $s_1$.

**Strdup** returns a pointer to a new string which is a duplicate of the string pointed to by $s_1$. The space for the new string is obtained using *malloc*(3C). If the new string can not be created, null is returned.

**Strncat** appends at most $n$ characters. Each returns a pointer to the null-terminated result.

**Strcmp** compares its arguments and returns an integer less than, equal to, or greater than 0, according as $s_1$ is lexicographically less than, equal to, or greater than $s_2$. *Strncmp* makes the same comparison but looks at at most $n$ characters.

**Strcpy** copies string $s_2$ to $s_1$, stopping after the null character has been copied. *Strncpy* copies exactly $n$ characters, truncating $s_2$ or adding null characters to $s_1$ if necessary. The result will not be null-terminated if the length of $s_2$ is $n$ or more. Each function returns $s_1$.

**Strlen** returns the number of characters in $s$, not including the terminating null character.

**Strchr** (*strrchr*) returns a pointer to the first (last) occurrence of character $c$ in string $s$, or a NULL pointer if $c$ does not occur in the string. The null character terminating a string is considered to be part of the string.

**Strpbrk** returns a pointer to the first occurrence in string $s_1$ of any character from string $s_2$, or a NULL pointer if no character from $s_2$ exists in $s_1$.

**Strspn** (*strcspn*) returns the length of the initial segment of string $s_1$ which consists entirely of characters from (not from) string $s_2$.

**Strtok** considers the string $s_1$ to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string $s_2$. The first call (with pointer $s_1$ specified) returns a pointer to the first character of the first token, and will have written a null character into $s_1$ immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument a NULL pointer) will work through the string $s_1$ immediately following that token. In this way subsequent calls will work through the string $s_1$ until no tokens remain. The separator string $s_2$ may be different from call to call. When no token remains in $s_1$, a NULL pointer is returned.

For user convenience, all these functions are declared in the optional *<string.h>* header file.

**SEE ALSO**

malloc(3C), malloc(3X).
CAVEATS

`strcmp` and `strncmp` are implemented by using the most natural character comparison on the machine. Thus the sign of the value returned when one of the characters has its high-order bit set not the same in all implementations and should not be relied upon.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.
NAME
strtod, atof — convert string to double-precision number

SYNOPSIS
double strtod (str, ptr)
char *str, **ptr;
double atof (str)
char *str;

DESCRIPTION
strtod returns as a double-precision floating-point number the value represented by the character string pointed to by str. The string is scanned up to the first unrecognized character.

strtod recognizes an optional string of "white-space" characters [as defined by isspace in ctype(3C)], then an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer.

If the value of ptr is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by ptr. If no number can be formed, *ptr is set to str, and zero is returned.

Atof(str) is equivalent to strtod(str, (char **)NULL).

SEE ALSO
ctype(3C), scanf(3S), strtol(3C).

DIAGNOSTICS
If the correct value would cause overflow, ±HUGE (as defined in <math.h>) is returned (according to the sign of the value), and errno is set to ERANGE.
If the correct value would cause underflow, zero is returned and errno is set to ERANGE.
NAME
strtol, atol, atoi — convert string to integer

SYNOPSIS

long strtol (str, ptr, base)
  char *str, **ptr;
  int base;

long atol (str)
  char *str;

int atoi (str)
  char *str;

DESCRIPTION

strtol returns as a long integer the value represented by the character string pointed to by str. The string is scanned up to the first character inconsistent with the base. Leading "white-space" characters [as defined by isspace in ctype(3C)] are ignored.

If the value of ptr is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by ptr. If no integer can be formed, that location is set to str, and zero is returned.

If base is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored, and "0x" or "OX" is ignored if base is 16.

If base is zero, the string itself determines the base thusly: After an optional leading sign a leading zero indicates octal conversion, and a leading "0x" or "0X" hexadecimal conversion. Otherwise, decimal conversion is used.

Truncation from long to int can, of course, take place upon assignment or by an explicit cast.

Atol(str) is equivalent to strtol(str, (char **)NULL, 10).

Atoi(str) is equivalent to (int) strtol(str, (char **)NULL, 10).

SEE ALSO
ctype(3C), scanf(3S), strtod(3C).

CAVEAT
Overflow conditions are ignored.
NAME
   swab — swap bytes

SYNOPSIS
   void swab (from, to, nbytes)
   char *from, *to;
   int nbytes;

DESCRIPTION
   swab copies nbytes bytes pointed to by from to the array pointed to by to, exchanging adjacent even and odd bytes. Nbytes should be even and non-negative. If nbytes is odd and positive swab uses nbytes–1 instead. If nbytes is negative, swab does nothing.
NAME
    system — issue a shell command

SYNOPSIS
    #include <stdio.h>
    int system (string)
    char *string;

DESCRIPTION
    system causes the string to be given to sh(1) as input, as if the string had been
typed as a command at a terminal. The current process waits until the shell has
completed, then returns the exit status of the shell.

FILES
    /bin/sh

SEE ALSO
    exec(2).

DIAGNOSTICS
    system forks to create a child process that in turn exec’s /bin/sh in order to exe-
cute string. If the fork or exec fails, system returns a negative value and sets errno.
NAME
tmpfile – create a temporary file

SYNOPSIS
#include <stdio.h>
FILE *tmpfile ()

DESCRIPTION
tmpfile creates a temporary file using a name generated by tmpnam(3S), and returns a corresponding FILE pointer. If the file cannot be opened, an error message is printed using perror(3C), and a NULL pointer is returned. The file will automatically be deleted when the process using it terminates. The file is opened for update ("w+").

SEE ALSO
creat(2), unlink(2), fopen(3S), mktemp(3C), perror(3C), stdio(3S), tmpnam(3S).
NAME
tmpnam, tempnam — create a name for a temporary file

SYNOPSIS

#include <stdio.h>
char *tmpnam (s)
char *s;
char *tempnam (dir, pfx)
char *dir, *pfx;

DESCRIPTION

These functions generate file names that can safely be used for a temporary file. 

tmpnam always generates a file name using the path-prefix defined as P tmpdir in the <stdio.h> header file. If s is NULL, tmpnam leaves its result in an internal static area and returns a pointer to that area. The next call to tmpnam will destroy the contents of the area. If s is not NULL, it is assumed to be the address of an array of at least L_tmpnam bytes, where L_tmpnam is a constant defined in <stdio.h>; tmpnam places its result in that array and returns s.

Tempnam allows the user to control the choice of a directory. The argument dir points to the name of the directory in which the file is to be created. If dir is NULL or points to a string that is not a name for an appropriate directory, the path-prefix defined as P tmpdir in the <stdio.h> header file is used. If that directory is not accessible, /tmp will be used as a last resort. This entire sequence can be up-staged by providing an environment variable TMPDIR in the user’s environment, whose value is the name of the desired temporary-file directory.

Many applications prefer their temporary files to have certain favorite initial letter sequences in their names. Use the pfx argument for this. This argument may be NULL or point to a string of up to five characters to be used as the first few characters of the temporary-file name.

Tempnam uses malloc(3C) to get space for the constructed file name, and returns a pointer to this area. Thus, any pointer value returned from tempnam may serve as an argument to free [see malloc(3C)]. If tempnam cannot return the expected result for any reason, i.e. malloc(3C) failed, or none of the above mentioned attempts to find an appropriate directory was successful, a NULL pointer will be returned.

NOTES

These functions generate a different file name each time they are called.

Files created using these functions and either fopen(3S) or creat(2) are temporary only in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the user’s responsibility to use unlink(2) to remove the file when its use is ended.

SEE ALSO

creat(2), unlink(2), fopen(3S), malloc(3C), mktemp(3C), tmpfile(3S).
CAVEATS

If called more than 17,576 times in a single process, these functions will start recycling previously used names.

Between the time a file name is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using these functions or mktemp, and the file names are chosen to render duplication by other means unlikely.
NAME
tsearch, tfind, tdelete, twalk — manage binary search trees

SYNOPSIS
#include <search.h>
char *tsearch ((char *)) key, (char **) rootp, compar)
int (*compar)();
char *tfind ((char *) key, (char **) rootp, compar)
int (*compar)();
char *tdelete ((char *) key, (char **) rootp, compar)
int (*compar)();
void twalk ((char *) root, action)
void (*action)();

DESCRIPTION
tsearch, tfind, tdelete, and twalk are routines for manipulating binary search trees. They are generalized from Knuth (6.2.2) Algorithms T and D. All comparisons are done with a user-supplied routine. This routine is called with two arguments, the pointers to the elements being compared. It returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.
tsearch is used to build and access the tree. Key is a pointer to a datum to be accessed or stored. If there is a datum in the tree equal to *key (the value pointed to by key), a pointer to this found datum is returned. Otherwise, *key is inserted, and a pointer to it returned. Only pointers are copied, so the calling routine must store the data. Rootp points to a variable that points to the root of the tree. A NULL value for the variable pointed to by rootp denotes an empty tree; in this case, the variable will be set to point to the datum which will be at the root of the new tree.

Like tsearch, tfind will search for a datum in the tree, returning a pointer to it if found. However, if it is not found, tfind will return a NULL pointer. The arguments for tfind are the same as for tsearch.

tdelete deletes a node from a binary search tree. The arguments are the same as for tsearch. The variable pointed to by rootp will be changed if the deleted node was the root of the tree. Tdelete returns a pointer to the parent of the deleted node, or a NULL pointer if the node is not found.

twalk traverses a binary search tree. Root is the root of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) Action is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument is the address of the node being visited. The second argument is a value from an enumeration data type typedef
enum { preorder, postorder, endorder, leaf } VISIT; (defined in the <search.h> header file), depending on whether this is the first, second or third time that the node has been visited (during a depth-first, left-to-right traversal of the tree), or
whether the node is a leaf. The third argument is the level of the node in the
tree, with the root being level zero.

The pointers to the key and the root of the tree should be of type pointer-to-
element, and cast to type pointer-to-character. Similarly, although declared as
type pointer-to-character, the value returned should be cast into type pointer-to-
element.

EXAMPLE
The following code reads in strings and stores structures containing a pointer to
each string and a count of its length. It then walks the tree, printing out the
stored strings and their lengths in alphabetical order.

```c
#include <search.h>
#include <stdio.h>

struct node {
    /* pointers to these are stored in the tree */
    char *string;
    int length;
};
char string_space[10000];    /* space to store strings */
struct node nodes[500];      /* nodes to store */
struct node *root = NULL;    /* this points to the root */

main( )
{
    char *strptr = string_space;
    struct node *nodeptr = nodes;
    void print_node( ), twalk( );
    int i = 0, node_compare( );

    while (gets(strptr) != NULL && i++ < 500) {
        /* set node */
        nodeptr->string = strptr;
        nodeptr->length = strlen(strptr);
        /* put node into the tree */
        (void) tsearch((char *)nodeptr, (char **) &root,
                       node_compare);
        /* adjust pointers, so we don't overwrite tree */
        strptr += nodeptr->length + 1;
        nodeptr++;
    }
    twalk((char *)root, print_node);
}
/*
This routine compares two nodes, based on an
alphabetical ordering of the string field.
```
/*
node_compare(node1, node2)
char *node1, *node2;
{
    return strcmp(((struct node *)node1)->string,
                   ((struct node *) node2)->string);
}
/*
This routine prints out a node, the first time
twalk encounters it.
*/
void
print_node(node, order, level)
char **node;
VISIT order;
int level;
{
    if (order == preorder || order == leaf) {
        (void)printf("string = %20s, length = %d\n",
                      (*((struct node **)node))->string,
                      (*((struct node **)node))->length);
    }
}

SEE ALSO
bsearch(3C), hsearch(3C), lsearch(3C).

DIAGNOSTICS
A NULL pointer is returned by tsearch if there is not enough space available to
create a new node.
A NULL pointer is returned by tfind and tdelete if rootp is NULL on entry.
If the datum is found, both tsearch and tfind return a pointer to it. If not, tfind
returns NULL, and tsearch returns a pointer to the inserted item.

WARNINGS
The root argument to twalk is one level of indirection less than the rootp argu-
ments to tsearch and tdelete.
There are two nomenclatures used to refer to the order in which tree nodes are
visited. tsearch uses preorder, postorder and endorder to respectively refer to
visiting a node before any of its children, after its left child and before its right,
and after both its children. The alternate nomenclature uses preorder, inorder
and postorder to refer to the same visits, which could result in some confusion
over the meaning of postorder.

CAVEAT
If the calling function alters the pointer to the root, results are unpredictable.
NAME
	ttyname, isatty -- find name of a terminal

SYNOPSIS
	char *ttyname (fildes)
	int fildes;
	n int isatty (fildes)
	int fildes;

DESCRIPTION

ttyname returns a pointer to a string containing the null-terminated path name of the terminal device associated with file descriptor fildes.

Isatty returns 1 if fildes is associated with a terminal device, 0 otherwise.

FILES
	/dev/*

DIAGNOSTICS

ttyname returns a NULL pointer if fildes does not describe a terminal device in directory /dev.

CAVEAT

The return value points to static data whose content is overwritten by each call.
NAME

ttyslot — find the slot in the utmp file of the current user

SYNOPSIS

int ttyslot ()

DESCRIPTION

ttyslot returns the index of the current user's entry in the /etc/utmp file. This is accomplished by actually scanning the file /etc/inittab for the name of the terminal associated with the standard input, the standard output, or the error output (0, 1 or 2).

FILES

/etc/inittab
/etc/utmp

SEE ALSO

gnutut(3C), ttyname(3C).

DIAGNOSTICS

A value of 0 is returned if an error was encountered while searching for the terminal name or if none of the above file descriptors is associated with a terminal device.
NAME
ungetc − push character back into input stream

SYNOPSIS
#include <stdio.h>
int ungetc (c, stream)
int c;
FILE *stream;

DESCRIPTION
ungetc inserts the character c into the buffer associated with an input stream. That character, c, will be returned by the next getc(3S) call on that stream. ungetc returns c, and leaves the file stream unchanged.

One character of pushback is guaranteed, provided something has already been read from the stream and the stream is actually buffered.

If c equals EOF, ungetc does nothing to the buffer and returns EOF.

Fseek(3S) erases all memory of inserted characters.

SEE ALSO
fseek(3S), getc(3S), setbuf(3S), stdio(3S).

DIAGNOSTICS
ungetc returns EOF if it cannot insert the character.

BUGS
When stream is stdin, one character may be pushed back onto the buffer without a previous read statement.
NAME
vprintf, vfprintf, vsprintf — print formatted output of a varargs argument list

SYNOPSIS
#include <stdio.h>
#include <varargs.h>
int vprintf (format, ap)
char *format;
va_list ap;
int vfprintf (stream, format, ap)
FILE *stream;
char *format;
va_list ap;
int vsprintf (s, format, ap)
char *s, *format;
va_list ap;

DESCRIPTION
vprintf, vfprintf, and vsprintf are the same as printf, fprintf, and sprintf respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by varargs(5).

EXAMPLE
The following demonstrates the use of vfprintf to write an error routine.
#include <stdio.h>
#include <varargs.h>

/*
* error should be called like
* error(function_name, format, arg1, arg2...); */
/*@VARARGS*/
void
error(va_list)
/*@ Note that the function_name and format arguments cannot be */
@ separately declared because of the definition of varargs. */
va_dcl
{
    va_list args;
    char *fmt;

    va_start(args);
    /* print out name of function causing error */
    (void)fprintf(stderr, "ERROR in %s: ", va_arg(args, char *));
    fmt = va_arg(args, char *);
/* print out remainder of message */
(void)vfprintf(stderr, fmt, args);
va_end(args);
(void)abort( );
}

SEE ALSO
printf(3S), varargs(5).
NAME
bessel: j0, j1, jn, y0, y1, yn — Bessel functions

SYNOPSIS
#include <math.h>

double j0 (x)
double x;
double j1 (x)
double x;
double jn (n, x)
int n;
double x;
double y0 (x)
double x;
double y1 (x)
double x;
double yn (n, x)
int n;
double x;

DESCRIPTION
j0 and j1 return Bessel functions of x of the first kind of orders 0 and 1 respectively. jn returns the Bessel function of x of the first kind of order n.

y0 and y1 return Bessel functions of x of the second kind of orders 0 and 1 respectively. yn returns the Bessel function of x of the second kind of order n. The value of x must be positive.

SEE ALSO
matherr(3M).

DIAGNOSTICS
Non-positive arguments cause y0, y1 and yn to return the value -HUGE and to set errno to EDOM. In addition, a message indicating DOMAIN error is printed on the standard error output.

Arguments too large in magnitude cause j0, j1, y0 and y1 to return zero and to set errno to ERANGE. In addition, a message indicating TLOSS error is printed on the standard error output.

These error-handling procedures may be changed with the function matherr(3M).
NAME
erf, erfc – error function and complementary error function

SYNOPSIS
#include <math.h>

double erf (x)

double erfc (x)

DESCRIPTION
erf returns the error function of x, defined as \( \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \).

erfc, which returns 1.0 − erf(x), is provided because of the extreme loss of relative accuracy if erf(x) is called for large x and the result subtracted from 1.0 (e.g., for x = 5, 12 places are lost).

SEE ALSO
exp(3M).
NAME
exp, log, log10, pow, sqrt — exponential, logarithm, power, square root functions

SYNOPSIS

#include <math.h>
double exp (x)
double x;
double log (x)
double x;
double log10 (x)
double x;
double pow (x, y)
double x, y;
double sqrt (x)
double x;

DESCRIPTION
exp returns \( e^x \).

Log returns the natural logarithm of \( x \). The value of \( x \) must be positive.

Log10 returns the logarithm base ten of \( x \). The value of \( x \) must be positive.

Pow returns \( x^y \). If \( x \) is zero, \( y \) must be positive. If \( x \) is negative, \( y \) must be an integer.

Sqrt returns the non-negative square root of \( x \). The value of \( x \) may not be negative.

SEE ALSO
hypot(3M), matherr(3M), sinh(3M).

DIAGNOSTICS
exp returns HUGE when the correct value would overflow, or 0 when the correct value would underflow, and sets errno to ERANGE.

Log and log10 return -HUGE and set errno to EDOM when \( x \) is non-positive. A message indicating DOMAIN error (or SING error when \( x \) is 0) is printed on the standard error output.

Pow returns 0 and sets errno to EDOM when \( x \) is 0 and \( y \) is non-positive, or when \( x \) is negative and \( y \) is not an integer. In these cases a message indicating DOMAIN error is printed on the standard error output. When the correct value for pow would overflow or underflow, pow returns ±HUGE or 0 respectively, and sets errno to ERANGE.

Sqrt returns 0 and sets errno to EDOM when \( x \) is negative. A message indicating DOMAIN error is printed on the standard error output.

These error-handling procedures may be changed with the function matherr(3M).
NAME
floor, ceil, fmod, fabs — floor, ceiling, remainder, absolute value functions

SYNOPSIS
#include <math.h>
double floor (x)
double x;
double ceil (x)
double x;
double fmod (x, y)
double x, y;
double fabs (x)
double x;

DESCRIPTION
floor returns the largest integer (as a double-precision number) not greater than x.

Ceil returns the smallest integer not less than x.

Fmod returns the floating-point remainder of the division of x by y: zero if y is zero or if x/y would overflow; otherwise the number f with the same sign as x, such that x = iy + f for some integer i, and |f| < |y|.

Fabs returns the absolute value of x, |x|.

SEE ALSO
abs(3C).
NAME
gamma — log gamma function

SYNOPSIS
#include <math.h>
double gamma (x)
double x;
extern int signgam;

DESCRIPTION

gamma returns ln(\Gamma(x)), where \Gamma(x) is defined as \int_0^\infty e^{-t}t^{x-1}dt. The sign of \Gamma(x) is returned in the external integer signgam. The argument x may not be a non-positive integer.

The following C program fragment might be used to calculate \Gamma:

  if ((y = gamma(x)) > LN_MAXDOUBLE)
    error();
  y = signgam * exp(y);

where LN_MAXDOUBLE is the least value that causes exp(3M) to return a range error, and is defined in the <values.h> header file.

SEE ALSO
exp(3M), matherr(3M), values(5).

DIAGNOSTICS
For non-negative integer arguments HUGE is returned, and errno is set to EDOM. A message indicating SING error is printed on the standard error output.

If the correct value would overflow, gamma returns HUGE and sets errno to ERANGE.

These error-handling procedures may be changed with the function matherr(3M).
NAME
hypot — Euclidean distance function

SYNOPSIS
#include <math.h>
double hypot (x, y)
double x, y;

DESCRIPTION
hypot returns

\sqrt{x \cdot x + y \cdot y},

taking precautions against unwarranted overflows.

SEE ALSO
matherr(3M).

DIAGNOSTICS
When the correct value would overflow, hypot returns HUGE and sets errno to ERANGE.

These error-handling procedures may be changed with the function matherr(3M).
NAME
matherr – error-handling function

SYNOPSIS
#include <math.h>
int matherr (x)
struct exception *x;

DESCRIPTION
matherr is invoked by functions in the Math Library when errors are detected. Users may define their own procedures for handling errors, by including a function named matherr in their programs. matherr must be of the form described above. When an error occurs, a pointer to the exception structure x will be passed to the user-supplied matherr function. This structure, which is defined in the <math.h> header file, is as follows:

struct exception {
    int type;
    char *name;
    double arg1, arg2, retval;
};

The element type is an integer describing the type of error that has occurred, from the following list of constants (defined in the header file):

DOMAIN argument domain error
SING argument singularity
OVERFLOW overflow range error
UNDERFLOW underflow range error
TLOSS total loss of significance
PLOSS partial loss of significance

The element name points to a string containing the name of the function that incurred the error. The variables arg1 and arg2 are the arguments with which the function was invoked. Retval is set to the default value that will be returned by the function unless the user’s matherr sets it to a different value.

If the user’s matherr function returns non-zero, no error message will be printed, and errno will not be set.

If matherr is not supplied by the user, the default error-handling procedures, described with the math functions involved, will be invoked upon error. These procedures are also summarized in the table below. In every case, errno is set to EDOM or ERANGE and the program continues.

EXAMPLE
#include <math.h>

int matherr(x)
register struct exception *x;
{
    switch (x->type) {
    case DOMAIN:
MATHERR(3M)  (Math Libraries)  MATHERR(3M)

/* change sqrt to return sqrt(-arg1), not 0 */
if (!strcmp(x->name, "sqrt")) {
    x->retval = sqrt(-x->arg1);
    return (0); /* print message and set errno */
}

case SING:
    /* all other domain or sing errors, print message and abort */
    fprintf(stderr, "domain error in %s\n", x->name);
    abort( );

case PLOSS:
    /* print detailed error message */
    fprintf(stderr, "loss of significance in %s(%g) = %g\n",
            x->name, x->arg1, x->retval);
    return (1); /* take no other action */
}

return (0); /* all other errors, execute default procedure */

<table>
<thead>
<tr>
<th>Type</th>
<th>Domain</th>
<th>Sing</th>
<th>OverFlow</th>
<th>UnderFlow</th>
<th>TLoss</th>
<th>PLoss</th>
</tr>
</thead>
<tbody>
<tr>
<td>BESS:</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>M,0</td>
<td>-</td>
</tr>
<tr>
<td>y0, y1, yn (arg ≤ 0)</td>
<td>M, -H</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EXP:</td>
<td>-</td>
<td>-</td>
<td>H</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LOG, LOG10:</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(arg &lt; 0)</td>
<td>M, -H</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(arg = 0)</td>
<td>-</td>
<td>M, -H</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>POW:</td>
<td>-</td>
<td>-</td>
<td>±H</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>neg ** non-int</td>
<td>M, 0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0 ** non-pos</td>
<td>-</td>
<td>-</td>
<td>±H</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SQRT:</td>
<td>M, 0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GAMMA:</td>
<td>-</td>
<td>M, H</td>
<td>H</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HYPOP:</td>
<td>-</td>
<td>-</td>
<td>H</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SINH:</td>
<td>-</td>
<td>-</td>
<td>±H</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>COSH:</td>
<td>-</td>
<td>-</td>
<td>H</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SIN, COS, TAN:</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>M, 0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ASIN, ACOS, ATAN2:</td>
<td>M, 0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

ABBREVIATIONS
- As much as possible of the value is returned.
  M Message is printed (EDOM error).
  H HUGE is returned.
  -H -HUGE is returned.
  ±H HUGE or -HUGE is returned.
  0 0 is returned.
NAME
sinh, cosh, tanh — hyperbolic functions

SYNOPSIS
#include <math.h>

double sinh (x)
double x;

double cosh (x)
double x;

double tanh (x)
double x;

DESCRIPTION
sinh, cosh, and tanh return, respectively, the hyperbolic sine, cosine and tangent
of their argument.

SEE ALSO
matherr(3M).

DIAGNOSTICS
sinh and cosh return HUGE (and sinh may return –HUGE for negative x) when
the correct value would overflow and set errno to ERANGE.

These error-handling procedures may be changed with the function matherr(3M).
NAME
trig: sin, cos, tan, asin, acos, atan, atan2 — trigonometric functions

SYNOPSIS
#include <math.h>
double sin (x)
double x;
double cos (x)
double x;
double tan (x)
double x;
double asin (x)
double x;
double acos (x)
double x;
double atan (x)
double x;
double atan2 (y, x)
double y, x;

DESCRIPTION
Sin, cos and tan return respectively the sine, cosine and tangent of their argument, x, measured in radians.

Asin returns the arcsine of x, in the range \([-\pi/2,\pi/2]\].

Acos returns the arccosine of x, in the range \([0,\pi]\].

Atan returns the arctangent of x, in the range \([-\pi/2,\pi/2]\].

Atan2 returns the arctangent of y/x, in the range \((-\pi,\pi]\), using the signs of both arguments to determine the quadrant of the return value.

SEE ALSO
matherr(3M).

DIAGNOSTICS
Sin, cos, and tan lose accuracy when their argument is far from zero. For arguments sufficiently large, these functions return zero when there would otherwise be a complete loss of significance. In this case a message indicating TLOSS error is printed on the standard error output. For less extreme arguments causing partial loss of significance, a PLOSS error is generated but no message is printed. In both cases, errno is set to ERANGE.

If the magnitude of the argument of asin or acos is greater than one, or if both arguments of atan2 are zero, zero is returned and errno is set to EDOM. In addition, a message indicating DOMAIN error is printed on the standard error output. These error-handling procedures may be changed with the function matherr(3M).
NAME
t_accept — accept a connect request

SYNOPSIS
#include <tiuser.h>
int t_accept(fd, resfd, call)
int fd;
int resfd;
struct t_call *call;

DESCRIPTION
This function is issued by a transport user to accept a connect request. 
FD identifies the local transport endpoint where the connect indication arrived, resfd 
specifies the local transport endpoint where the connection is to be established, 
and call contains information required by the transport provider to complete the 
connection. Call points to a t_call structure which contains the following 
members:

struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;

Netbuf is described in intro(3). In call, addr is the address of the caller, opt indicates any protocol-specific parameters associated with the connection, udata points to any user data to be returned to the caller, and sequence is the value returned by t_listen that uniquely associates the response with a previously received connect indication.

A transport user may accept a connection on either the same, or on a different, 
local transport endpoint than the one on which the connect indication arrived. If 
the same endpoint is specified (i.e., resfd=fd), the connection can be accepted 
unless the following condition is true: The user has received other indications on 
that endpoint but has not responded to them (with t_accept or t_snddis). For this 
condition, t_accept will fail and set t_errno to TBADF.

If a different transport endpoint is specified (resfd!=fd), the endpoint must be 
bound to a protocol address and must be in the T_IDLE state [see t_getstate(3N)] 
before the t_accept is issued.

For both types of endpoints, t_accept will fail and set t_errno to TLOOK if there are indications (e.g., a connect or disconnect) waiting to be received on that end-
point.

The values of parameters specified by opt and the syntax of those values are 
protocol specific. The udata argument enables the called transport user to send 
user data to the caller and the amount of user data must not exceed the limits 
supported by the transport provider as returned by t_open or t_getinfo. If the len 
[see netbuf in intro(3)] field of udata is zero, no data will be sent to the caller.

On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport 
endpoint, or the user is illegally accepting a connection on

369
T_ACCEPT(3N) (Networking Support Utilities) T_ACCEPT(3N)

the same transport endpoint on which the connect indication arrived.

[TOUTSTATE] The function was issued in the wrong sequence on the transport endpoint referenced by fd, or the transport endpoint referred to by resfd is not in the T_IDLE state.

[TACCESS] The user does not have permission to accept a connection on the responding transport endpoint or use the specified options.

[TBADOPT] The specified options were in an incorrect format or contained illegal information.

[TBADDATA] The amount of user data specified was not within the bounds allowed by the transport provider.

[TBADSEQ] An invalid sequence number was specified.

[TLOOK] An asynchronous event has occurred on the transport endpoint referenced by fd and requires immediate attention.

[TNOTSUPPORT] This function is not supported by the underlying transport provider.

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO intro(3), t_connect(3N), t_getstate(3N), t_listen(3N), t_open(3N), t_rcvconnect(3N).


DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate the error.
NAME
t_alloc — allocate a library structure

SYNOPSIS
#include <tiuser.h>
char *t_alloc(fd, struct_type, fields)
int fd;
int struct_type;
int fields;

DESCRIPTION
The t_alloc function dynamically allocates memory for the various transport
function argument structures as specified below. This function will allocate
memory for the specified structure, and will also allocate memory for buffers
referenced by the structure.

The structure to allocate is specified by struct_type, and can be one of the fol­
lowing:
T_BIND struct t_bind
T_CALL struct t_call
T_OPTMGMT struct t_optmgmt
T_DIS struct t_discon
T_UNITDATA struct t_unitdata
T_UDERROR struct t_uderr
T_INFO struct t_info

where each of these structures may subsequently be used as an argument to one
or more transport functions.

Each of the above structures, except T_INFO, contains at least one field of type
struct netbuf. Netbuf is described in intro(3). For each field of this type, the user
may specify that the buffer for that field should be allocated as well. The fields
argument specifies this option, where the argument is the bitwise-OR of any of
the following:
T_ADDR The addr field of the t_bind, t_call, t_unitdata, or t_uderr structures.
T_OPT The opt field of the t_optmgmt, t_call, t_unitdata, or t_uderr structures.
TADATA The udata field of the t_call, t_discon, or t_unitdata structures.
T_ALL All relevant fields of the given structure.

For each field specified in fields, t_alloc will allocate memory for the buffer asso­
ciated with the field, and initialize the buf pointer and maxlen [see netbuf in
intro(3) for description of buf and maxlen] field accordingly. The length of the
buffer allocated will be based on the same size information that is returned to
the user on t_open and t_getinfo. Thus, fd must refer to the transport endpoint
through which the newly allocated structure will be passed, so that the
appropriate size information can be accessed. If the size value associated with
any specified field is -1 or -2 (see t_open or t_getinfo), t_alloc will be unable to
determine the size of the buffer to allocate and will fail, setting t_errno to
TSYSERR and errno to EINV. For any field not specified in fields, buf will be set to NULL and maxlen will be set to zero.

Use of t_alloc to allocate structures will help ensure the compatibility of user programs with future releases of the transport interface.

On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO
intro(3), t_free(3N), t_getinfo(3N), t_open(3N).

DIAGNOSTICS
On successful completion, t_alloc returns a pointer to the newly allocated structure. On failure, NULL is returned.
NAME
    t_bind — bind an address to a transport endpoint

SYNOPSIS
    #include <tiuser.h>
    int t_bind(fd, req, ret)
    int fd;
    struct t_bind *req;
    struct t_bind *ret;

DESCRIPTION
    This function associates a protocol address with the transport endpoint specified
    by fd and activates that transport endpoint. In connection mode, the transport
    provider may begin accepting or requesting connections on the transport end­
    point. In connectionless mode, the transport user may send or receive data units
    through the transport endpoint.

    The req and ret arguments point to a t_bind structure containing the following
    members:

    struct netbuf addr;
    unsigned qlen;

    Netbuf is described in intro(3). The addr field of the t_bind structure specifies a
    protocol address and the qlen field is used to indicate the maximum number of
    outstanding connect indications.

    Req is used to request that an address, represented by the netbuf structure, be
    bound to the given transport endpoint. Len [see netbuf in intro(3); also for buf
    and maxlen] specifies the number of bytes in the address and buf points to the
    address buffer. Maxlen has no meaning for the req argument. On return, ret
    contains the address that the transport provider actually bound to the transport
    endpoint; this may be different from the address specified by the user in req. In
    ret, the user specifies maxlen which is the maximum size of the address buffer
    and buf which points to the buffer where the address is to be placed. On return,
    len specifies the number of bytes in the bound address and buf points to the
    bound address. If maxlen is not large enough to hold the returned address, an
    error will result.

    If the requested address is not available, or if no address is specified in req (the
    len field of addr in req is zero) the transport provider will assign an appropriate
    address to be bound, and will return that address in the addr field of ret. The
    user can compare the addresses in req and ret to determine whether the trans­
    port provider bound the transport endpoint to a different address than that
    requested.

    Req may be NULL if the user does not wish to specify an address to be bound. Here,
    the value of qlen is assumed to be zero, and the transport provider must
    assign an address to the transport endpoint. Similarly, ret may be NULL if the
    user does not care what address was bound by the provider and is not interested
    in the negotiated value of qlen. It is valid to set req and ret to NULL for the
    same call, in which case the provider chooses the address to bind to the trans­
    port endpoint and does not return that information to the user.

373
The `qlen` field has meaning only when initializing a connection-mode service. It specifies the number of outstanding connect indications the transport provider should support for the given transport endpoint. An outstanding connect indication is one that has been passed to the transport user by the transport provider. A value of `qlen` greater than zero is only meaningful when issued by a passive transport user that expects other users to call it. The value of `qlen` will be negotiated by the transport provider and may be changed if the transport provider cannot support the specified number of outstanding connect indications. On return, the `qlen` field in `ret` will contain the negotiated value.

This function allows more than one transport endpoint to be bound to the same protocol address (however, the transport provider must support this capability also), but it is not allowable to bind more than one protocol address to the same transport endpoint. If a user binds more than one transport endpoint to the same protocol address, only one endpoint can be used to listen for connect indications associated with that protocol address. In other words, only one `t_bind` for a given protocol address may specify a value of `qlen` greater than zero. In this way, the transport provider can identify which transport endpoint should be notified of an incoming connect indication. If a user attempts to bind a protocol address to a second transport endpoint with a value of `qlen` greater than zero, the transport provider will assign another address to be bound to that endpoint. If a user accepts a connection on the transport endpoint that is being used as the listening endpoint, the bound protocol address will be found to be busy for the duration of that connection. No other transport endpoints may be bound for listening while that initial listening endpoint is in the data transfer phase. This will prevent more than one transport endpoint bound to the same protocol address from accepting connect indications.

On failure, `t_errno` may be set to one of the following:

- `[TBADF]` The specified file descriptor does not refer to a transport endpoint.
- `[TOUTSTATE]` The function was issued in the wrong sequence.
- `[TBADADDR]` The specified protocol address was in an incorrect format or contained illegal information.
- `[TNOADDR]` The transport provider could not allocate an address.
- `[TACCES]` The user does not have permission to use the specified address.
- `[TBUFOVFLW]` The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The provider's state will change to `T_IDLE` and the information to be returned in `ret` will be discarded.
- `[T SYSERR]` A system error has occurred during execution of this function.
SEE ALSO

intro(3), t_open(3N), t_optmgmt(3N), t_unbind(3N).


DIAGNOSTICS

t_bind returns 0 on success and -1 on failure and t_errno is set to indicate the error.
NAME
t_close — close a transport endpoint

SYNOPSIS
#include <tiuser.h>

int t_close(fd)
int fd;

DESCRIPTION
The t_close function informs the transport provider that the user is finished with
the transport endpoint specified by fd, and frees any local library resources associ­
ated with the endpoint. In addition, t_close closes the file associated with the
transport endpoint.

t_close should be called from the T_UNBND state [see t_getstate (3N)]. However,
this function does not check state information, so it may be called from any state
to close a transport endpoint. If this occurs, the local library resources associ­
ated with the endpoint will be freed automatically. In addition, close(2) will be
issued for that file descriptor; the close will be abortive if no other process has
that file open, and will break any transport connection that may be associated
with that endpoint.

On failure, t_errno may be set to the following:
[TBADF] The specified file descriptor does not refer to a transport endpoint.

SEE ALSO
t_getstate(3N), t_open(3N), t_unbind(3N).

DIAGNOSTICS
t_close returns 0 on success and -1 on failure and t_errno is set to indicate the
error.
NAME
t_connect — establish a connection with another transport user

SYNOPSIS

```
#include <tiuser.h>

int t_connect(fd, sndcall, rcvcall)
    int fd;
    struct t_call *sndcall;
    struct t_call *rcvcall;
```

DESCRIPTION

This function enables a transport user to request a connection to the specified
destination transport user. *Fd* identifies the local transport endpoint where com­
munication will be established, while *sndcall* and *rcvcall* point to a *t_call* struc­
ture which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

*Sndcall* specifies information needed by the transport provider to establish a con­
nection and *rcvcall* specifies information that is associated with the newly esta­
blished connection.

*Netbuf* is described in *intro(3)*. In *sndcall*, *addr* specifies the protocol address of
the destination transport user, *opt* presents any protocol-specific information that
might be needed by the transport provider, *udata* points to optional user data
that may be passed to the destination transport user during connection establish­
ment, and *sequence* has no meaning for this function.

On return in *rcvcall*, *addr* returns the protocol address associated with the
responding transport endpoint, *opt* presents any protocol-specific information
associated with the connection, *udata* points to optional user data that may be
returned by the destination transport user during connection establishment, and
*sequence* has no meaning for this function.

The *opt* argument implies no structure on the options that may be passed to the
transport provider. The transport provider is free to specify the structure of any
options passed to it. These options are specific to the underlying protocol of the
transport provider. The user may choose not to negotiate protocol options by
setting the *len* field of *opt* to zero. In this case, the provider may use default
options.

The *udata* argument enables the caller to pass user data to the destination tran­
sport user and receive user data from the destination user during connection
establishment. However, the amount of user data must not exceed the limits
supported by the transport provider as returned by *t_open(3N)* or *t_getinfo(3N)*.
If the *len* [see *netbuf* in *intro(3)*] field of *udata* is zero in *sndcall*, no data will be
sent to the destination transport user.

On return, the *addr*, *opt*, and *udata* fields of *rcvcall* will be updated to reflect
values associated with the connection. Thus, the *maxlen* [see *netbuf* in *intro(3)*]
field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, \texttt{rcvcall} may be NULL, in which case no information is given to the user on return from \texttt{t_connect}.

By default, \texttt{t_connect} executes in synchronous mode, and will wait for the destination user's response before returning control to the local user. A successful return (i.e. return value of zero) indicates that the requested connection has been established. However, if \texttt{O_NDELAY} is set (via \texttt{t_open} or \texttt{fcntl}), \texttt{t_connect} executes in asynchronous mode. In this case, the call will not wait for the remote user's response, but will return control immediately to the local user and return -1 with \texttt{t_errno} set to \texttt{TNODATA} to indicate that the connection has not yet been established. In this way, the function simply initiates the connection establishment procedure by sending a connect request to the destination transport user.

On failure, \texttt{t_errno} may be set to one of the following:

- [TBADF] The specified file descriptor does not refer to a transport endpoint.
- [TOUTSTATE] The function was issued in the wrong sequence.
- [TNODATA] \texttt{O_NDELAY} was set, so the function successfully initiated the connection establishment procedure, but did not wait for a response from the remote user.
- [TBADADDR] The specified protocol address was in an incorrect format or contained illegal information.
- [TBAOPT] The specified protocol options were in an incorrect format or contained illegal information.
- [TBADDATA] The amount of user data specified was not within the bounds allowed by the transport provider.
- [TACCES] The user does not have permission to use the specified address or options.
- [TBUFOVFLW] The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. If executed in synchronous mode, the provider's state, as seen by the user, changes to \texttt{T_DATAXFER}, and the connect indication information to be returned in \texttt{rcvcall} is discarded.
- [TLOOK] An asynchronous event has occurred on this transport endpoint and requires immediate attention.
- [TNOTSUPPORT] This function is not supported by the underlying transport provider.
- [TSYSERR] A system error has occurred during execution of this function.

\textbf{SEE ALSO}

\texttt{intro(3), t_accept(3N), t_getinfo(3N), t_listen(3N), t_open(3N), t_optmgmt(3N), t_rcvconnect(3N)}.

DIAGNOSTICS

\emph{t_connect} returns 0 on success and -1 on failure and \texttt{t_errno} is set to indicate the error.
NAME
t_error — produce error message

SYNOPSIS
#include <tiuser.h>
void t_error(errmsg)
  char *errmsg;
extern int t_errno;
extern char *t_errlist[];
extern int t_nerr;

DESCRIPTION
The t_error function produces a message on the standard error output which
describes the last error encountered during a call to a transport function. The
argument string errmsg is a user-supplied error message that gives context to the
error. t_error prints the user-supplied error message followed by a colon and a
standard error message for the current error defined in t_errno. To simplify
variant formatting of messages, the array of message strings t_errlist is provided;
t_errno can be used as an index in this table to get the message string without
the newline. T_nerr is the largest message number provided for in the t_errlist
table.

T_errno is only set when an error occurs and is not cleared on successful calls.

EXAMPLE
If a t_connect function fails on transport endpoint fd2 because a bad address was
given, the following call might follow the failure:

t_error("t_connect failed on fd2");

The diagnostic message to be printed would look like:

t_connect failed on fd2: Incorrect transport address format

where "Incorrect transport address format" identifies the specific error that
occurred, and "t_connect failed on fd2" tells the user which function failed on
which transport endpoint.

SEE ALSO
NAME
t_free — free a library structure

SYNOPSIS
#include <tiuser.h>

int t_free(ptr, struct_type)
char *ptr;
int struct_type;

DESCRIPTION
The t_free function frees memory previously allocated by t_alloc. This function
will free memory for the specified structure, and will also free memory for
buffers referenced by the structure.

Ptr points to one of the six structure types described for t_alloc, and struct_type
identifies the type of that structure which can be one of the following:
T_BIND struct t_bind
T_CALL struct t_call
T_OPTMGM struct t_optmgmt
T_DIS struct t_discon
T_UNITDATA struct t_unitdata
T_UDERROR struct t_uderr
T_INFO struct t_info

where each of these structures is used as an argument to one or more transport
functions.

t_free will check the addr, opt, and udata fields of the given structure (as
appropriate), and free the buffers pointed to by the buf field of the netbuf [see
intro(3)] structure. If buf is NULL, t_free will not attempt to free memory. After
all buffers are freed, t_free will free the memory associated with the structure
pointed to by ptr.

Undefined results will occur if ptr or any of the buf pointers points to a block of
memory that was not previously allocated by t_alloc.

On failure, t_errno may be set to the following:
[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO
intro(3), t_alloc(3N).

DIAGNOSTICS
t_free returns 0 on success and -1 on failure and t_errno is set to indicate the
error.
NAME

t_getinfo - get protocol-specific service information

SYNOPSIS

#include <tiuser.h>

int t_getinfo(fd, info)
int fd;
struct t_info *info;

DESCRIPTION

This function returns the current characteristics of the underlying transport protocol associated with file descriptor fd. The info structure is used to return the same information returned by t_open. This function enables a transport user to access this information during any phase of communication.

This argument points to a t_info structure which contains the following members:

long addr;  /* max size of the transport protocol address */
long options;  /* max number of bytes of protocol-specific options */
long tsdu;  /* max size of a transport service data unit (TSDU) */
long etsdu;  /* max size of an expedited transport service data unit (ETSDU) */
long connect;  /* max amount of data allowed on connection establishment functions */
long discon;  /* max amount of data allowed on t_snddis and t_rcvdis functions */
long servtype;  /* service type supported by the transport provider */

The values of the fields have the following meanings:

addr
A value greater than or equal to zero indicates the maximum size of a transport protocol address; a value of -1 specifies that there is no limit on the address size; and a value of -2 specifies that the transport provider does not provide user access to transport protocol addresses.

options
A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the provider; a value of -1 specifies that there is no limit on the option size; and a value of -2 specifies that the transport provider does not support user-settable options.

tsdu
A value greater than zero specifies the maximum size of a transport service data unit (TSDU); a value of zero specifies that the transport provider does not support the concept of TSDU, although it does support the sending of a data stream with no logical boundaries preserved across a connection; a value of -1 specifies that there is no limit on the size of a TSDU; and a value of -2 specifies that the transfer of normal data is not supported by the transport provider.

etsdu
A value greater than zero specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited
data stream with no logical boundaries preserved across a connection; a value of -1 specifies that there is no limit on the size of an ETSDU; and a value of -2 specifies that the transfer of expedited data is not supported by the transport provider.

**connect**
A value greater than or equal to zero specifies the maximum amount of data that may be associated with connection establishment functions; a value of -1 specifies that there is no limit on the amount of data sent during connection establishment; and a value of -2 specifies that the transport provider does not allow data to be sent with connection establishment functions.

**discon**
A value greater than or equal to zero specifies the maximum amount of data that may be associated with the `t_snddis` and `t_rvcdis` functions; a value of -1 specifies that there is no limit on the amount of data sent with these abortive release functions; and a value of -2 specifies that the transport provider does not allow data to be sent with the abortive release functions.

**servtype**
This field specifies the service type supported by the transport provider, as described below.

If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the `t_alloc` function may be used to allocate these buffers. An error will result if a transport user exceeds the allowed data size on any function. The value of each field may change as a result of option negotiation, and `t_getinfo` enables a user to retrieve the current characteristics.

The `servtype` field of `info` may specify one of the following values on return:

**T_COTS**
The transport provider supports a connection-mode service but does not support the optional orderly release facility.

**T_COTS_ORD**
The transport provider supports a connection-mode service with the optional orderly release facility.

**T_CLTS**
The transport provider supports a connectionless-mode service. For this service type, `t_open` will return -2 for `etsdu`, `connect`, and `discon`.

On failure, `t_errno` may be set to one of the following:

**[TBADF]**
The specified file descriptor does not refer to a transport endpoint.

**[TSYSERR]**
A system error has occurred during execution of this function.

See Also
`t_open(3N)`.

T_GETINFO(3N) (Networking Support Utilities) T_GETINFO(3N)

DIAGNOSTICS

t_getinfo returns 0 on success and -1 on failure and t_errno is set to indicate the error.
T_GETSTATE(3N) (Networking Support Utilities) T_GETSTATE(3N)

NAME
t_getstate — get the current state

SYNOPSIS
#include <tiuser.h>
int t_getstate(fd)
int fd;

DESCRIPTION
The t_getstate function returns the current state of the provider associated with
the transport endpoint specified by fd.

On failure, t_errno may be set to one of the following:
[TBADF] The specified file descriptor does not refer to a transport endpoint.
[TSTATECHNG] The transport provider is undergoing a state change.
[TSYSERR] A system error has occurred during execution of this func-
tion.

SEE ALSO
t_open(3N).

DIAGNOSTICS
The t_getstate returns the current state on successful completion and -1 on failure and
t_errno is set to indicate the error. The current state may be one of the fol-
lowing:

T_UNBND unused
T_IDLE idle
T_OUTCON outgoing connection pending
T_INCON incoming connection pending
T_DATAFER data transfer
T_OUTREL outgoing orderly release (waiting for an orderly release indica-
tion)
T_INREL incoming orderly release (waiting for an orderly release request)

If the provider is undergoing a state transition when t_getstate is called, the func-
tion will fail.
NAME

t_learn — listen for a connect request

SYNOPSIS

#include <tiuser.h>

int t_listen(fd, call)
int fd;
struct t_call *call;

DESCRIPTION

This function listens for a connect request from a calling transport user. Fd
identifies the local transport endpoint where connect indications arrive, and on
return, call contains information describing the connect indication. Call points to
a t_call structure which contains the following members:

struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;

Netbuf is described in intro(3). In call, addr returns the protocol address of the
calling transport user, opt returns protocol-specific parameters associated with
the connect request, udata returns any user data sent by the caller on the con­
nect request, and sequence is a number that uniquely identifies the returned con­
nect indication. The value of sequence enables the user to listen for multiple
connect indications before responding to any of them.

Since this function returns values for the addr, opt, and udata fields of call, the
maxlen [see netbuf in intro(3)] field of each must be set before issuing the t_listen
to indicate the maximum size of the buffer for each.

By default, t_listen executes in synchronous mode and waits for a connect indi­
cation to arrive before returning to the user. However, if O_NDELAY is set (via
t_open or fcntl), t_listen executes asynchronously, reducing to a poll for existing
connect indications. If none are available, it returns -1 and sets t_errno to
TNODATA.

On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport
endpoint.

[TBUFOVFLW] The number of bytes allocated for an incoming argument
is not sufficient to store the value of that argument. The
provider’s state, as seen by the user, changes to T_INCON,
and the connect indication information to be returned in
call is discarded.

[TNODATA] O_NDELAY was set, but no connect indications had been
queued.

[TLOOK] An asynchronous event has occurred on this transport
endpoint and requires immediate attention.
T_LISTEN(3N) (Networking Support Utilities) T_LISTEN(3N)

[TNOTSUPPORT] This function is not supported by the underlying transport provider.

[TSYSERR] A system error has occurred during execution of this function.

CAVEATS
If a user issues \texttt{t\_listen} in synchronous mode on a transport endpoint that was not bound for listening (i.e. \texttt{qlen} was zero on \texttt{t\_bind}), the call will wait forever because no connect indications will arrive on that endpoint.

SEE ALSO
\texttt{intro(3), t\_accept(3N), t\_bind(3N), t\_connect(3N), t\_open(3N), t\_rcvconnect(3N)}.
\textit{Network Programmer's Guide}.

DIAGNOSTICS
\texttt{t\_listen} returns 0 on success and -1 on failure and \texttt{t\_errno} is set to indicate the error.
NAME
t_look — look at the current event on a transport endpoint

SYNOPSIS
#include <tiuser.h>
int t_look(fd)
int fd;

DESCRIPTION
This function returns the current event on the transport endpoint specified by fd. This function enables a transport provider to notify a transport user of an asynchronous event when the user is issuing functions in synchronous mode. Certain events require immediate notification of the user and are indicated by a specific error, TLOOK, on the current or next function to be executed.

This function also enables a transport user to poll a transport endpoint periodically for asynchronous events.

On failure, t_errno may be set to one of the following:
[TBADF] The specified file descriptor does not refer to a transport endpoint.
[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO
t_open(3N), Network Programmer's Guide.

DIAGNOSTICS
Upon success, t_look returns a value that indicates which of the allowable events has occurred, or returns zero if no event exists. One of the following events is returned:
T_LISTEN connection indication received
T_CONNECT connect confirmation received
T_DATA normal data received
T_EXDATA expedited data received
T_DISCONNECT disconnect received
T_ERROR fatal error indication
T_UDERR datagram error indication
T_ORDREL orderly release indication

On failure, -1 is returned and t_errno is set to indicate the error.
NAME
t_open – establish a transport endpoint

SYNOPSIS
#include <tiuser.h>

int t_open(path, oflag, info)
char *path;
int oflag;
struct t_info *info;

DESCRIPTION
t_open must be called as the first step in the initialization of a transport endpoint. This function establishes a transport endpoint by opening a UNIX file that identifies a particular transport provider (i.e., transport protocol) and returning a file descriptor that identifies that endpoint. For example, opening the file /dev/iso_cots identifies an OSI connection-oriented transport layer protocol as the transport provider.

Path points to the path name of the file to open, and oflag identifies any open flags [as in open(2)]. t_open returns a file descriptor that will be used by all subsequent functions to identify the particular local transport endpoint.

This function also returns various default characteristics of the underlying transport protocol by setting fields in the t_info structure. This argument points to a t_info which contains the following members:

long addr; /* max size of the transport protocol address */
long options; /* max number of bytes of protocol-specific options */
long tsdu; /* max size of a transport service data unit (TSDU) */
long etsdu; /* max size of an expedited transport service data unit (ETSDU) */
long connect; /* max amount of data allowed on connection establishment functions */
long discon; /* max amount of data allowed on t_snddis and t_rcvdis functions */
long servtype; /* service type supported by the transport provider */

The values of the fields have the following meanings:
addr
A value greater than or equal to zero indicates the maximum size of a transport protocol address; a value of -1 specifies that there is no limit on the address size; and a value of -2 specifies that the transport provider does not provide user access to transport protocol addresses.

options
A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the provider; a value of -1 specifies that there is no limit on the option size; and a value of -2 specifies that the transport provider does not support user-settable options.

tsdu
A value greater than zero specifies the maximum size of a transport service data unit (TSDU); a value of zero specifies that the
transport provider does not support the concept of TSDU, although it does support the sending of a data stream with no logical boundaries preserved across a connection; a value of -1 specifies that there is no limit on the size of a TSDU; and a value of -2 specifies that the transfer of normal data is not supported by the transport provider.

etsdu A value greater than zero specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of -1 specifies that there is no limit on the size of an ETSDU; and a value of -2 specifies that the transfer of expedited data is not supported by the transport provider.

connect A value greater than or equal to zero specifies the maximum amount of data that may be associated with connection establishment functions; a value of -1 specifies that there is no limit on the amount of data sent during connection establishment; and a value of -2 specifies that the transport provider does not allow data to be sent with connection establishment functions.

discon A value greater than or equal to zero specifies the maximum amount of data that may be associated with the t_snddis and t_rcvdis functions; a value of -1 specifies that there is no limit on the amount of data sent with these abortive release functions; and a value of -2 specifies that the transport provider does not allow data to be sent with the abortive release functions.

serotype This field specifies the service type supported by the transport provider, as described below.

If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the t_alloc function may be used to allocate these buffers. An error will result if a transport user exceeds the allowed data size on any function.

The serotype field of info may specify one of the following values on return:

T_COTS The transport provider supports a connection-mode service but does not support the optional orderly release facility.

T_COTS_ORD The transport provider supports a connection-mode service with the optional orderly release facility.

T_CLTS The transport provider supports a connectionless-mode service. For this service type, t_open will return -2 for etsdu, connect, and discon.

A single transport endpoint may support only one of the above services at one time.
If `info` is set to NULL by the transport user, no protocol information is returned by `t_open`.

On failure, `t_errno` may be set to the following:

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO 
`open(2)`,

DIAGNOSTICS 
`t_open` returns a valid file descriptor on success and -1 on failure and `t_errno` is set to indicate the error.
NAME
t_optmgmt – manage options for a transport endpoint

SYNOPSIS
#include <tiuser.h>
int t_optmgmt(fd, req, ret)
int fd;
struct t_optmgmt *req;
struct t_optmgmt *ret;

DESCRIPTION
The t_optmgmt function enables a transport user to retrieve, verify, or negotiate
protocol options with the transport provider. Fd identifies a bound transport
endpoint.

The req and ret arguments point to a t_optmgmt structure containing the fol­
lowing members:

    struct netbuf opt;
    long flags;

The opt field identifies protocol options and the flags field is used to specify the
action to take with those options.

The options are represented by a netbuf [see intro(3); also for len, buf and maxlen]  
structure in a manner similar to the address in t_bind. Req is used to request a
specific action of the provider and to send options to the provider. Len specifies
the number of bytes in the options, buf points to the options buffer, and maxlen
has no meaning for the req argument. The transport provider may return
options and flag values to the user through ret. For ret, maxlen specifies the
maximum size of the options buffer and buf points to the buffer where the
options are to be placed. On return, len specifies the number of bytes of options
returned. Maxlen has no meaning for the req argument, but must be set in the
ret argument to specify the maximum number of bytes the options buffer can
hold. The actual structure and content of the options is imposed by the tran­
sport provider.

The flags field of req can specify one of the following actions:

T_NEGOTIATE This action enables the user to negotiate the values of the
options specified in req with the transport provider. The pro­
vider will evaluate the requested options and negotiate the
values, returning the negotiated values through ret.

T_CHECK This action enables the user to verify whether the options
specified in req are supported by the transport provider. On
return, the flags field of ret will have either T_SUCCESS or
T_FAILURE set to indicate to the user whether the options are
supported. These flags are only meaningful for the T_CHECK
request.
This action enables a user to retrieve the default options supported by the transport provider into the opt field of ret. In req, the len field of opt must be zero and the buf field may be NULL.

If issued as part of the connectionless-mode service, t_optmgmt may block due to flow control constraints. The function will not complete until the transport provider has processed all previously sent data units.

On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TOUTSTATE] The function was issued in the wrong sequence.

[TACCESS] The user does not have permission to negotiate the specified options.

[TBADOPT] The specified protocol options were in an incorrect format or contained illegal information.

[TBADFLAG] An invalid flag was specified.

[TBUFOVFLW] The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The information to be returned in ret will be discarded.

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO
intro(3), t_getinfo(3N), t_open(3N).


DIAGNOSTICS

_t_optmgmt returns 0 on success and -1 on failure and _t_errno is set to indicate the error.
NAME
  t_rcv — receive data or expedited data sent over a connection

SYNOPSIS
  int t_rcv(fd, buf, nbytes, flags)
  int fd;
  char *buf;
  unsigned nbytes;
  int *flags;

DESCRIPTION
This function receives either normal or expedited data. Fd identifies the local
transport endpoint through which data will arrive, buf points to a receive buffer
where user data will be placed, and nbytes specifies the size of the receive buffer.
Flags may be set on return from t_rcv and specifies optional flags as described
below.

By default, t_rcv operates in synchronous mode and will wait for data to arrive if
none is currently available. However, if O_NDELAY is set (via t_open or fcntl),
t_rcv will execute in asynchronous mode and will fail if no data is available.
(See TNODATA below.)

On return from the call, if T_MORE is set in flags this indicates that there is more
data and the current transport service data unit (TSDU) or expedited transport
service data unit (ETSDU) must be received in multiple t_rcv calls. Each t_rcv
with the T_MORE flag set indicates that another t_rcv must follow immediately to
get more data for the current TSDU. The end of the TSDU is identified by the
return of a t_rcv call with the T_MORE flag not set. If the transport provider
does not support the concept of a TSDU as indicated in the info argument on
return from t_open or t_getinfo, the T_MORE flag is not meaningful and should be
ignored.

On return, the data returned is expedited data if T_EXPEDITED is set in flags. If
the number of bytes of expedited data exceeds nbytes, t_rcv will set T_EXPEDITED
and T_MORE on return from the initial call. Subsequent calls to retrieve the
remaining ETSDU will not have T_EXPEDITED set on return. The end of the
ETSDU is identified by the return of a t_rcv call with the T_MORE flag not set.

If expedited data arrives after part of a TSDU has been retrieved, receipt of the
remainder of the TSDU will be suspended until the ETSDU has been processed.
Only after the full ETSDU has been retrieved (T_MORE not set) will the
remainder of the TSDU be available to the user.

On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport
        endpoint.

[TNODATA] O_NDELAY was set, but no data is currently available from
          the transport provider.

[TLOOK] An asynchronous event has occurred on this transport end-
        point and requires immediate attention.

[TNOTSUPPORT] This function is not supported by the underlying transport
              provider.
A system error has occurred during execution of this function.

**SEE ALSO**
- `t_open(3N), t_snd(3N)`.

**DIAGNOSTICS**

On successful completion, `t_rcv` returns the number of bytes received, and it returns -1 on failure and `t_errno` is set to indicate the error.
NAME

\texttt{t_rccvconnect} — receive the confirmation from a connect request

SYNOPSIS

\begin{verbatim}
#include \langle tiuser.h \rangle
int t_rccvconnect(fd, call)
int fd;
struct t_call *call;
\end{verbatim}

DESCRIPTION

This function enables a calling transport user to determine the status of a previously sent connect request and is used in conjunction with \texttt{t_connect} to establish a connection in asynchronous mode. The connection will be established on successful completion of this function.

\textit{Fd} identifies the local transport endpoint where communication will be established, and \textit{call} contains information associated with the newly established connection. \textit{Call} points to a \texttt{t_call} structure which contains the following members:

\begin{verbatim}
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
\end{verbatim}

\textit{Netbuf} is described in \texttt{intro(3)}. In \textit{call}, \textit{addr} returns the protocol address associated with the responding transport endpoint, \textit{opt} presents any protocol-specific information associated with the connection, \textit{udata} points to optional user data that may be returned by the destination transport user during connection establishment, and \textit{sequence} has no meaning for this function.

The \textit{maxlen} [see \texttt{netbuf} in \texttt{intro(3)}] field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, \textit{call} may be \texttt{NULL}, in which case no information is given to the user on return from \texttt{t_rccvconnect}. By default, \texttt{t_rccvconnect} executes in synchronous mode and waits for the connection to be established before returning. On return, the \textit{addr}, \textit{opt}, and \textit{udata} fields reflect values associated with the connection.

If \texttt{O_NDELAY} is set (via \texttt{t_open} or \texttt{fcntl}), \texttt{t_rccvconnect} executes in asynchronous mode, and reduces to a poll for existing connect confirmations. If none are available, \texttt{t_rccvconnect} fails and returns immediately without waiting for the connection to be established. (See \texttt{TNODATA} below.) \texttt{t_rccvconnect} must be re-issued at a later time to complete the connection establishment phase and retrieve the information returned in \textit{call}.

On failure, \texttt{t_errno} may be set to one of the following:

\begin{itemize}
\item [\texttt{TBADF}] The specified file descriptor does not refer to a transport endpoint.
\item [\texttt{TBUFOVFLW}] The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument and the connect information to be returned in \textit{call} will be discarded. The provider's state, as seen by the user, will be changed to \texttt{DATAXFER}.
\end{itemize}
T_RXCONNECT(3N) (Networking Support Utilities) T_RXCONNECT(3N)

[TNODATA] O_NDELAY was set, but a connect confirmation has not yet arrived.

[TLOOK] An asynchronous event has occurred on this transport connection and requires immediate attention.

[TNOTSUPPORT] This function is not supported by the underlying transport provider.

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO
intro(3), t_accept(3N), t_bind(3N), t_connect(3N), t_listen(3N), t_open(3N).

DIAGNOSTICS
$ t_rxconnect$ returns 0 on success and -1 on failure and $t_errno$ is set to indicate the error.
NAME
t_rcvdis — retrieve information from disconnect

SYNOPSIS
#include <tiuser.h>

t_rcvdis(fd, discon)
int fd;
struct t_discon *discon;

DESCRIPTION
This function is used to identify the cause of a disconnect, and to retrieve any
user data sent with the disconnect. Fd identifies the local transport endpoint
where the connection existed, and discon points to a t_discon structure containing
the following members:

    struct netbuf udata;
    int reason;
    int sequence;

Netbuf is described in intro(3). Reason specifies the reason for the disconnect
through a protocol-dependent reason code, udata identifies any user data that
was sent with the disconnect, and sequence may identify an outstanding connect
indication with which the disconnect is associated. Sequence is only meaningful
when t_rcvdis is issued by a passive transport user who has executed one or
more t_listen functions and is processing the resulting connect indications. If a
disconnect indication occurs, sequence can be used to identify which of the out-
standing connect indications is associated with the disconnect.

If a user does not care if there is incoming data and does not need to know the
value of reason or sequence, discon may be NULL and any user data associated
with the disconnect will be discarded. However, if a user has retrieved more
than one outstanding connect indication (via t_listen) and discon is NULL, the
user will be unable to identify with which connect indication the disconnect is
associated.

On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport
        endpoint.

[TNODIS] No disconnect indication currently exists on the specified
        transport endpoint.

[TBUFOVFLW] The number of bytes allocated for incoming data is not
            sufficient to store the data. The provider's state, as seen
            by the user, will change to T_IDLE, and the disconnect
            indication information to be returned in discon will be dis-
            carded.

[TNOTSUPPORT] This function is not supported by the underlying transport
              provider.

[TSYSERR] A system error has occurred during execution of this func-
          tion.
T_RCVDIS(3N)  (Networking Support Utilities)  T_RCVDIS(3N)

SEE ALSO
   intro(3), t_connect(3N), t_listen(3N), t_open(3N), t_snddis(3N).

DIAGNOSTICS
   t_rcvdis returns 0 on success and -1 on failure and t_errno is set to indicate the error.
NAME
	t_rcvrel — acknowledge receipt of an orderly release indication

SYNOPSIS

#include <tiuser.h>

t_rcvrel(fd)

int fd;

DESCRIPTION

This function is used to acknowledge receipt of an orderly release indication. Fd identifies the local transport endpoint where the connection exists. After receipt of this indication, the user may not attempt to receive more data because such an attempt will block forever. However, the user may continue to send data over the connection if t_sndrel has not been issued by the user.

This function is an optional service of the transport provider, and is only supported if the transport provider returned service type T_COTS_ORD on t_open or t_getinfo.

On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TNOREL] No orderly release indication currently exists on the specified transport endpoint.

[TLOOK] An asynchronous event has occurred on this transport endpoint and requires immediate attention.

[TNOTSUPPORT] This function is not supported by the underlying transport provider.

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO

t_open(3N), t_sndrel(3N),

DIAGNOSTICS

t_rcvrel returns 0 on success and -1 on failure t_errno is set to indicate the error.
NAME
t_rcvudata — receive a data unit

SYNOPSIS
#include <tiuser.h>
int t_rcvudata(fd, unitdata, flags)
int fd;
struct t_unitdata *unitdata;
int *flags;

DESCRIPTION
This function is used in connectionless mode to receive a data unit from another
transport user. Fd identifies the local transport endpoint through which data will
be received, unitdata holds information associated with the received data unit,
and flags is set on return to indicate that the complete data unit was not
received. Unitdata points to a t_unitdata structure containing the following
members:

struct netbuf addr;
struct netbuf opt;
struct netbuf udata;

The maxlen [see netbuf in intro(3)] field of addr, opt, and udata must be set before
issuing this function to indicate the maximum size of the buffer for each.

On return from this call, addr specifies the protocol address of the sending user,
opt identifies protocol-specific options that were associated with this data unit,
and udata specifies the user data that was received.

By default, t_rcvudata operates in synchronous mode and will wait for a data
unit to arrive if none is currently available. However, if O_NDELAY is set (via
t_open or fcntl), t_rcvudata will execute in asynchronous mode and will fail if no
data units are available.

If the buffer defined in the udata field of unitdata is not large enough to hold the
current data unit, the buffer will be filled and T_MORE will be set in flags on
return to indicate that another t_rcvudata should be issued to retrieve the rest of
the data unit. Subsequent t_rcvudata call(s) will return zero for the length of the
address and options until the full data unit has been received.

On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TNODATA] O_NDELAY was set, but no data units are currently available
from the transport provider.

[TBUFOVFLW] The number of bytes allocated for the incoming protocol
address or options is not sufficient to store the information. The unit data information to be returned in unitdata
will be discarded.

[TLOOK] An asynchronous event has occurred on this transport endpoint and requires immediate attention.
**T_RCVUDATA(3N)** (Networking Support Utilities)

**TNOTSUPPORT**
This function is not supported by the underlying transport provider.

**TSYSERR**
A system error has occurred during execution of this function.

**SEE ALSO**
intro(3), t_rcvuderr(3N), t_sndudata(3N).

**DIAGNOSTICS**
*t_rcvudata* returns 0 on successful completion and -1 on failure and *t_errno* is set to indicate the error.
NAME
t_rcvuderr — receive a unit data error indication
SYNOPSIS
#include <tiuser.h>
int t_rcvuderr(fd, uderr)
    int fd;
    struct t_uderr *uderr;
DESCRIPTION
This function is used in connectionless mode to receive information concerning
an error on a previously sent data unit, and should only be issued following a
unit data error indication. It informs the transport user that a data unit with a
specific destination address and protocol options produced an error. Fd identifies
the local transport endpoint through which the error report will be received, and
uderr points to a t_uderr structure containing the following members:
    struct netbuf addr;
    struct netbuf opt;
    long error;
Netbuf is described in intro(3). The maxlen [see netbuf in intro(3)] field of addr
and opt must be set before issuing this function to indicate the maximum size of
the buffer for each.
On return from this call, the addr structure specifies the destination protocol
address of the erroneous data unit, the opt structure identifies protocol-specific
options that were associated with the data unit, and error specifies a protocol-
dependent error code.
If the user does not care to identify the data unit that produced an error, uderr
may be set to NULL and t_rcvuderr will simply clear the error indication without
reporting any information to the user.
On failure, t_errno may be set to one of the following:
[TBADF] The specified file descriptor does not refer to a transport endpoint.
[TNOUDERR] No unit data error indication currently exists on the specified
transport endpoint.
[TBUFOVFLW] The number of bytes allocated for the incoming protocol
address or options is not sufficient to store the information.
The unit data error information to be returned in uderr will be discarded.
[TNOTSUPPORT] This function is not supported by the underlying transport provider.
[TSYSERR] A system error has occurred during execution of this function.
SEE ALSO
intro(3), t_rcvudata(3N), t_sndudata(3N).

DIAGNOSTICS

_t_rcvuderr returns 0 on successful completion and -1 on failure and _errno is set to indicate the error.
NAME

t_snd — send data or expedited data over a connection

SYNOPSIS

```c
#include <tiuser.h>

int t_snd(fd, buf, nbytes, flags)

int fd;
char *buf;
unsigned nbytes;
int flags;
```

DESCRIPTION

This function is used to send either normal or expedited data. `fd` identifies the local transport endpoint over which data should be sent, `buf` points to the user data, `nbytes` specifies the number of bytes of user data to be sent, and `flags` specifies any optional flags described below.

By default, `t_snd` operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if O_NDELAY is set (via `t_open` or `fcntl`), `t_snd` will execute in asynchronous mode, and will fail immediately if there are flow control restrictions.

On successful completion, `t_snd` returns the number of bytes accepted by the transport provider. Normally this will equal the number of bytes specified in `nbytes`. However, if O_NDELAY is set, it is possible that only part of the data will be accepted by the transport provider. In this case, `t_snd` will set T_MORE for the data that was sent (see below) and will return a value less than `nbytes`. If `nbytes` is zero, no data will be passed to the provider and `t_snd` will return zero.

If T_EXPEDITED is set in `flags`, the data will be sent as expedited data, and will be subject to the interpretations of the transport provider.

If T_MORE is set in `flags`, or set as described above, an indication is sent to the transport provider that the transport service data unit (TSDU) (or expedited transport service data unit - ETSDU) is being sent through multiple `t_snd` calls. Each `t_snd` with the T_MORE flag set indicates that another `t_snd` will follow with more data for the current TSDU. The end of the TSDU (or ETSDU) is identified by a `t_snd` call with the T_MORE flag not set. Use of T_MORE enables a user to break up large logical data units without losing the boundaries of those units at the other end of the connection. The flag implies nothing about how the data is packaged for transfer below the transport interface. If the transport provider does not support the concept of a TSDU as indicated in the info argument on return from `t_open` or `t_getinfo`, the T_MORE flag is not meaningful and should be ignored.

The size of each TSDU or ETSDU must not exceed the limits of the transport provider as returned by `t_open` or `t_getinfo`. Failure to comply will result in protocol error EPROTO. (See TSYSERR below.)

If `t_snd` is issued from the T_IDLE state, the provider may silently discard the data. If `t_snd` is issued from any state other than T_DATAEXFER or T_IDLE, the provider will generate an EPROTO error.

405
On failure, \texttt{t_errno} may be set to one of the following:

- \texttt{[TBADF]}: The specified file descriptor does not refer to a transport endpoint.
- \texttt{[TFLOW]}: \texttt{O_NDELAY} was set, but the flow control mechanism prevented the transport provider from accepting data at this time.
- \texttt{[TNOTSUPPORT]}: This function is not supported by the underlying transport provider.
- \texttt{[TSYSERR]}: A system error has occurred during execution of this function.

\textbf{SEE ALSO}
\begin{itemize}
\item \texttt{t_open(3N)}, \texttt{t_rcv(3N)}.
\item \textit{Network Programmer's Guide}.
\end{itemize}

\textbf{DIAGNOSTICS}

On successful completion, \texttt{t_snd} returns the number of bytes accepted by the transport provider, and it returns -1 on failure and \texttt{t_errno} is set to indicate the error.
NAME
t_snddis — send user-initiated disconnect request

SYNOPSIS
#include <tiuser.h>
int t_snddis(fd, call)
int fd;
struct t_call *call;

DESCRIPTION
This function is used to initiate an abortive release on an already established
collection or to reject a connect request. Fd identifies the local transport end-
point of the connection, and call specifies information associated with the abort-
tive release. Call points to a t_call structure which contains the following
members:

    struct netbuf addr;
    struct netbuf opt;
    struct netbuf udata;
    int sequence;

Netbuf is described in intro(3). The values in call have different semantics,
depending on the context of the call to t_snddis. When rejecting a connect
request, call must be non-NULL and contain a valid value of sequence to
uniquely identify the rejected connect indication to the transport provider. The
addr and opt fields of call are ignored. In all other cases, call need only be used
when data is being sent with the disconnect request. The addr, opt, and
sequence fields of the t_call structure are ignored. If the user does not wish to
send data to the remote user, the value of call may be NULL.

Udata specifies the user data to be sent to the remote user. The amount of user
data must not exceed the limits supported by the transport provider as returned
by t_open or t_getinfo. If the len field of udata is zero, no data will be sent to the
remote user.

On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport
endpoint.

[TOUTSTATE] The function was issued in the wrong sequence. The trans-
port provider’s outgoing queue may be flushed, so data
may be lost.

[TBADDATA] The amount of user data specified was not within the
bounds allowed by the transport provider. The transport
provider’s outgoing queue will be flushed, so data may be
lost.

[TBADSEQ] An invalid sequence number was specified, or a NULL call
structure was specified when rejecting a connect request.
The transport provider’s outgoing queue will be flushed, so
data may be lost.
[TLOOK] An asynchronous event has occurred on this transport end-
point and requires immediate attention.

[TNOTSUPPORT] This function is not supported by the underlying transport
provider.

[TSYSERR] A system error has occurred during execution of this func-
tion.

SEE ALSO
intro(3), t_connect(3N), t_getinfo(3N), t_listen(3N), t_open(3N).

DIAGNOSTICS

t_snddis returns 0 on success and -1 on failure and t_errno is set to indicate the
error.
NAME
  t_sndrel — initiate an orderly release

SYNOPSIS
  #include <tiuser.h>
  int t_sndrel(fd)
  int fd;

DESCRIPTION
  This function is used to initiate an orderly release of a transport connection and
  indicates to the transport provider that the transport user has no more data to
  send. *Fd* identifies the local transport endpoint where the connection exists.
  After issuing *t_sndrel*, the user may not send any more data over the connection.
  However, a user may continue to receive data if an orderly release indication has
  been received.

  This function is an optional service of the transport provider, and is only sup­
  ported if the transport provider returned service type T_COTS_ORD on *t_open* or
  *t_getinfo*.

  On failure, *t_errno* may be set to one of the following:

  [TBADF] The specified file descriptor does not refer to a transport
          endpoint.

  [TFLOW] O_NDELAY was set, but the flow control mechanism
          prevented the transport provider from accepting the func­
          tion at this time.

  [TNOTSUPPORT] This function is not supported by the underlying transport
                provider.

  [TSYSERR] A system error has occurred during execution of this func­
            tion.

SEE ALSO
  *t_open*(3N), *t_rvrel*(3N).
  *Network Programmer's Guide*.

DIAGNOSTICS
  *t_sndrel* returns 0 on success and -1 on failure and *t_errno* is set to indicate the
  error.
NAME
t_sndudata — send a data unit

SYNOPSIS
#include <tiuser.h>
int t_sndudata(fd, unitdata)
int fd;
struct t_unitdata *unitdata;

DESCRIPTION
This function is used in connectionless mode to send a data unit to another trans-
port user. Fd identifies the local transport endpoint through which data will be
sent, and unitdata points to a t_unitdata structure containing the following
members:

struct netbuf addr;
struct netbuf opt;
struct netbuf udata;

Netbuf is described in intro(3). In unitdata, addr specifies the protocol address of
the destination user, opt identifies protocol-specific options that the user wants
associated with this request, and udata specifies the user data to be sent. The
user may choose not to specify what protocol options are associated with the
transfer by setting the len field of opt to zero. In this case, the provider may use
default options.

If the len field of udata is zero, no data unit will be passed to the transport pro-
vider; t_sndudata will not send zero-length data units.

By default, t_sndudata operates in synchronous mode and may wait if flow con-
tral restrictions prevent the data from being accepted by the local transport pro-
vider at the time the call is made. However, if O_NDELAY is set (via t_open or
fcntl), t_sndudata will execute in asynchronous mode and will fail under such
conditions.

If t_sndudata is issued from an invalid state, or if the amount of data specified in
udata exceeds the TSDU size as returned by t_open or t_getinfo, the provider will
generate an EPROTO protocol error. (See TSYSERR below.)

On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport
endpoint.

[TFLOW] O_NDELAY was set, but the flow control mechanism
prevented the transport provider from accepting data at this
time.

[TNOTSUPPORT] This function is not supported by the underlying transport
provider.

[TSYSERR] A system error has occurred during execution of this func-
tion.
T_SNDUDATA(3N) (Networking Support Utilities)  

SEE ALSO
intro(3), t_rcvudata(3N), t_rcvuderr(3N).

DIAGNOSTICS
T_sndudata returns 0 on successful completion and -1 on failure t_errno is set to indicate the error.
NAME
t_sync — synchronize transport library

SYNOPSIS

#include <tiuser.h>

int t_sync(fd)
int fd;

DESCRIPTION

For the transport endpoint specified by fd, t_sync synchronizes the data structures managed by the transport library with information from the underlying transport provider. In doing so, it can convert a raw file descriptor [obtained via open(2), dup(2), or as a result of a fork(2) and exec(2)] to an initialized transport endpoint, assuming that file descriptor referenced a transport provider. This function also allows two cooperating processes to synchronize their interaction with a transport provider.

For example, if a process forks a new process and issues an exec, the new process must issue a t_sync to build the private library data structure associated with a transport endpoint and to synchronize the data structure with the relevant provider information.

It is important to remember that the transport provider treats all users of a transport endpoint as a single user. If multiple processes are using the same endpoint, they should coordinate their activities so as not to violate the state of the provider. t_sync returns the current state of the provider to the user, thereby enabling the user to verify the state before taking further action. This coordination is only valid among cooperating processes; it is possible that a process or an incoming event could change the provider’s state after a t_sync is issued.

If the provider is undergoing a state transition when t_sync is called, the function will fail. On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor is a valid open file descriptor but does not refer to a transport endpoint.
[TSTATECHNG] The transport provider is undergoing a state change.
[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO
dup(2), exec(2), fork(2), open(2).

DIAGNOSTICS

t_sync returns the state of the transport provider on successful completion and -1 on failure and t_errno is set to indicate the error. The state returned may be one of the following:

T_UNBND unbound
T_IDLE idle
<table>
<thead>
<tr>
<th>T_SYNC(3N)</th>
<th>(Networking Support Utilities)</th>
<th>T_SYNC(3N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_OUTCON</td>
<td>outgoing connection pending</td>
<td></td>
</tr>
<tr>
<td>T_INCON</td>
<td>incoming connection pending</td>
<td></td>
</tr>
<tr>
<td>T_DATAFER</td>
<td>data transfer</td>
<td></td>
</tr>
<tr>
<td>T_OUTREL</td>
<td>outgoing orderly release (waiting for an orderly release indication)</td>
<td></td>
</tr>
<tr>
<td>T_INREL</td>
<td>incoming orderly release (waiting for an orderly release request)</td>
<td></td>
</tr>
</tbody>
</table>
T_UNBIND(3N) (Networking Support Utilities) T_UNBIND(3N)

NAME
t_unbind — disable a transport endpoint

SYNOPSIS

#include <tiuser.h>

int t_unbind(fd)
int fd;

DESCRIPTION

The t_unbind function disables the transport endpoint specified by fd which was previously bound by t_bind (3N). On completion of this call, no further data or events destined for this transport endpoint will be accepted by the transport provider.

On failure, t_errno may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TOUTSTATE] The function was issued in the wrong sequence.

[TLOOK] An asynchronous event has occurred on this transport endpoint.

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO

t_bind(3N).

DIAGNOSTICS

t_unbind returns 0 on success and -1 on failure and t_errno is set to indicate the error.
NAME
assert — verify program assertion

SYNOPSIS
#include <assert.h>
assert (expression)
int expression;

DESCRIPTION
This macro is useful for putting diagnostics into programs. When it is executed, if expression is false (zero), assert prints

"Assertion failed: expression, file xyz, line nnn"

on the standard error output and aborts. In the error message, xyz is the name of the source file and nnn the source line number of the assert statement.

Compiling with the preprocessor option \-DNDEBUG [see cpp(1)], or with the preprocessor control statement \#define NDEBUG ahead of the \#include <assert.h>" statement, will stop assertions from being compiled into the program.

SEE ALSO
cpp(1), abort(3C).

CAVEAT
Since assert is implemented as a macro, the expression may not contain any string literals.
NAME
crypt – password and file encryption functions

SYNOPSIS
cc [flag ...] file ... -lcrypt
char *crypt (key, salt)
char *key, *salt;
void setkey (key)
char *key;
void encrypt (block, flag)
char *block;
int flag;
char *des_crypt (key, salt)
char *key, *salt;
void des_setkey (key)
char *key;
void des_encrypt (block, flag)
char *block;
int flag;
int run_setkey (p, key)
int p[2];
char *key;
int run_crypt (offset, buffer, count, p)
long offset;
char *buffer;
unsigned int count;
int p[2];
int crypt_close(p)
int p[2];

DESCRIPTION
des_crypt is the password encryption function. It is based on a one way hashing
encryption algorithm with variations intended (among other things) to frustrate
use of hardware implementations of a key search.

Key is a user’s typed password. Salt is a two-character string chosen from the
set [a-zA-Z0-9.:]; this string is used to perturb the hashing algorithm in one of
4096 different ways, after which the password is used as the key to encrypt
repeatedly a constant string. The returned value points to the encrypted pass­
word. The first two characters are the salt itself.

The des_setkey and des_encrypt entries provide (rather primitive) access to the
actual hashing algorithm. The argument of des_setkey is a character array of
length 64 containing only the characters with numerical value 0 and 1. If this
string is divided into groups of 8, the low-order bit in each group is ignored; this
gives a 56-bit key which is set into the machine. This is the key that will be
used with the hashing algorithm to encrypt the string block with the function
des_encrypt.
The argument to the des_encrypt entry is a character array of length 64 containing only the characters with numerical value 0 and 1. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the hashing algorithm using the key set by des_setkey. If edflag is zero, the argument is encrypted; if non-zero, it is decrypted.

Note that decryption is not provided in the international version of crypt(3X). The international version is part of the C Programming Language Utilities, and the domestic version is part of the Security Administration Utilities. If decryption is attempted with the international version of des_encrypt, an error message is printed.

Crypt, setkey, and encrypt are front-end routines that invoke des_crypt, des_setkey, and des_encrypt respectively.

The routines run_setkey and run_crypt are designed for use by applications that need cryptographic capabilities [such as ed(1) and vi(1)] that must be compatible with the crypt(1) user-level utility. Run_setkey establishes a two-way pipe connection with crypt(1), using key as the password argument. Run_crypt takes a block of characters and transforms the cleartext or ciphertext into their ciphertext or cleartext using crypt(1). Offset is the relative byte position from the beginning of the file that the block of text provided in block is coming from. Count is the number of characters in block, and connection is an array containing indices to a table of input and output file streams. When encryption is finished, crypt_close is used to terminate the connection with crypt(1).

Run_setkey returns -1 if a connection with crypt(1) cannot be established. This will occur on international versions of UNIX where crypt(1) is not available. If a null key is passed to run_setkey, 0 is returned. Otherwise, 1 is returned. Run_crypt returns -1 if it cannot write output or read input from the pipe attached to crypt. Otherwise it returns 0.

DIAGNOSTICS

In the international version of crypt(3X), a flag argument of 1 to des_encrypt is not accepted, and an error message is printed.

SEE ALSO

getpass(3C), passwd(4).

CAVEAT

The return value in crypt points to static data that are overwritten by each call.
NAME

curses – terminal screen handling and optimization package

SYNOPSIS

The *curses* manual page is organized as follows:

In **SYNOPSIS**
- compiling information
- summary of parameters used by *curses* routines
- alphabetical list of *curses* routines, showing their parameters

In **DESCRIPTION**:
- An overview of how *curses* routines should be used

In **ROUTINES**, descriptions of each *curses* routines, are grouped under the appropriate topics:
- Overall Screen Manipulation
- Window and Pad Manipulation
- Output
- Input
- Output Options Setting
- Input Options Setting
- Environment Queries
- Soft Labels
- Low-level *curses* Access
- Terminfo-Level Manipulations
- Termcap Emulation
- Miscellaneous
- Use of *curscr*

Then come sections on:
- ATTRIBUTES
- FUNCTION CALLS
- LINE GRAPHICS

`cc [flag ...] file ... -lcurses [library ...]`

#include `<curses.h>` (automatically includes `<stdio.h>`, `<termio.h>`, and `<unctrl.h>`).

The parameters in the following list are not global variables, but rather this is a summary of the parameters used by the *curses* library routines. All routines return the int values ERR or OK unless otherwise noted. Routines that return pointers always return NULL on error. (ERR, OK, and NULL are all defined in `<curses.h>`.) Routines that return integers are not listed in the parameter list below.

`bool` `bf`

`char` `*area, *boolnames[], *boolcodes[], *boolfnames[], *bp`

`char` `*cap, *capname, codename[2], erasechar, *filename, *fmt`
CURSES(3X)
(Terminal Information Utilities)

char *keyname, killchar, *label, *longname
char *name, *numnames[], *numcodes[], *numfnames[]
char *slk_label, *str, *strnames[], *strcodes[], *strfnames[]

ctype attr, ch, horch, vertch
FILE *infd, *outfd
int begin_x, begin_y, begline, bot, c, col, count
int dmaxcol, dmaxrow, dmincol, dminrow, *errret, fildes
int (*init( )), labfmt, labnum, line
int ms, ncols, new, newcol, newrow, nlines, numlines
int oldcol, oldrow, overlay
int p1, p2, p9, pmincol, pminrow, (*putc( )), row
int smaxcol, smaxrow, smincol, sminrow, start
int tenths, top, visibility, x, y
SCREEN *new, *newterm, *set_term
TERMINAL *cur_term, *nterm, *oterm
va_list varglist

addch(ch)
addstr(str)
attroff(attrs)
attron(attrs)
attrset(attrs)
baudrate()
beep()
box(win, vertch, horch)
cbreak()
clear()
clearok(win, bf)
cmrtobot()}
clruoeol()
copywin(srcwin, dstwin, sminrow, smincol, dminrow, dmincol,
        dmaxrow, dmaxcol, overlay)"
curs_set(visibility)
def_prog_mode()
def_shell_mode()
del_curterm(oterm)
delay_output(ms)
delch()}
deleteln()}
delwin(win)
doupdate()
draino(ms)
echo()}
echochar(ch)endwin()
erases()
erasechar()
filter()
flash()
flushinp()
garbage_lines(win, begline, numlines)
getbegyx(win, y, x)
getch()
getmaxyx(win, y, x)
getstr(str)
getsyx(y, x)
getyx(win, y, x)
halfdelay(tenths)
has_ic()
has_il()
idlok(win, bf)
inch()
initscr()
insch(ch)
insertln()
intrflush(win, bf)
isendwin()
keyname(c)
keypad(win, bf)
killchar()
leaveok(win, bf)
longname()
meta(win, bf)
move(y, x)
mvaddch(y, x, ch)
mvaddstr(y, x, str)
mvcurs(oldrow, oldcol, newrow, newcol)
mvdelch(y, x)
mvgetch(y, x)
mvgetstr(y, x, str)
mvinch(y, x)
mvinsch(y, x, ch)
mvprintw(y, x, fmt [, arg...])
mvscanw(y, x, fmt [, arg...])
mvwaddch(win, y, x, ch)
mvwaddstr(win, y, x, str)
mvwdelch(win, y, x)
mvwgetch(win, y, x)
mvwgetstr(win, y, x, str)
mvwin(win, y, x)
mvwinch(win, y, x)
mvwinsch(win, y, x, ch)
mvwprintw(win, y, x, fmt [, arg...])
mvwscanw(win, y, x, fmt [, arg...])
napms(ms)
newpad(nlines, ncols)
newterm(type, outfd, infd)
newwin(nlines, ncols, begin_y, begin_x)
nl()
nocbreak()
nodelay(win, bf)
noecho()
onl()
noraw()
notimeout(win, bf)
overlay(srcwin, dstwin)
overwrite(srcwin, dstwin)
pechochar(pad, ch)
pnoutrefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)
prefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)
printw(fmt [, arg...])
putp(str)
raw()
refresh()
reset_prog_mode()
reset_shell_mode()
resetty()
restartterm(term, fildes, errret)
ripoffline(line, init)
savetty()
scanw(fmt [, arg...])
scr_dump(filename)
scr_init(filename)
scr_restore(filename)
scroll(win)
scrollok(win, bf)
set_curterm(nterm)
set_term(new)
setscreg(top, bot)
setyx(y, x)
setupterm(term, fildes, errret)
slk_clear()
slk_init(fmt)
slk_label(labnum)
slk_noutrefresh()
slk_refresh()
slk_restore()
slk_set(labnum, label, fmt)
slk_touch()
standend()
standout()
subpad(orig, nlines, ncols, begin_y, begin_x)
subwin(orig, nlines, ncols, begin_y, begin_x)
DESCRIPTION
The *curses* routines give the user a terminal-independent method of updating screens with reasonable optimization.

422
In order to initialize the routines, the routine `initscr()` or `newterm()` must be called before any of the other routines that deal with windows and screens are used. (Three exceptions are noted where they apply.) The routine `endwin()` must be called before exiting. To get character-at-a-time input without echoing, (most interactive, screen oriented programs want this) after calling `initscr()` you should call "cbreak(); noecho();" Most programs would additionally call "nonl(); intrflush (stdscr, FALSE); keypad(stdscr, TRUE);".

Before a `curses` program is run, a terminal’s tab stops should be set and its initialization strings, if defined, must be output. This can be done by executing the `tput init` command after the shell environment variable `TERM` has been exported. For further details, see `profile(4)`, `tput(l)`, and the "Tabs and Initialization" subsection of `terminfo(4)`.

The `curses` library contains routines that manipulate data structures called `windows` that can be thought of as two-dimensional arrays of characters representing all or part of a terminal screen. A default window called `stdscr` is supplied, which is the size of the terminal screen. Others may be created with `newwin()`. Windows are referred to by variables declared as `WINDOW *`; the type `WINDOW` is defined in `<curses.h>` to be a C structure. These data structures are manipulated with routines described below, among which the most basic are `move()` and `addch()`. (More general versions of these routines are included with names beginning with `w`, allowing you to specify a window. The routines not beginning with `w` usually affect `stdscr`.) Then `refresh()` is called, telling the routines to make the user’s terminal screen look like `stdscr`. The characters in a window are actually of type `chttype`, so that other information about the character may also be stored with each character.

Special windows called `pads` may also be manipulated. These are windows which are not constrained to the size of the screen and whose contents need not be displayed completely. See the description of `newpad()` under "Window and Pad Manipulation" for more information.

In addition to drawing characters on the screen, video attributes may be included which cause the characters to show up in modes such as underlined or in reverse video on terminals that support such display enhancements. Line drawing characters may be specified to be output. On input, `curses` is also able to translate arrow and function keys that transmit escape sequences into single values. The video attributes, line drawing characters, and input values use names, defined in `<curses.h>`, such as `A_REVERSE`, `ACS_HLINE`, and `KEY_LEFT`.

`curses` also defines the `WINDOW *` variable, `curscr`, which is used only for certain low-level operations like clearing and redrawing a garbaged screen. `curscr` can be used in only a few routines. If the window argument to `clearok()` is `curscr`, the next call to `wrefresh()` with any window will cause the screen to be cleared and repainted from scratch. If the window argument to `wrefresh()` is `curscr`, the screen in immediately cleared and repainted from scratch. This is how most programs would implement a "repaint-screen" function. More information on using `curscr` is provided where its use is appropriate.

The environment variables `LINES` and `COLUMNS` may be set to override `terminfo`'s idea of how large a screen is. These may be used in an AT&T Teletype 5620 layer, for example, where the size of a screen is changeable.
If the environment variable TERMINFO is defined, any program using curses will check for a local terminal definition before checking in the standard place. For example, if the environment variable TERM is set to att4425, then the compiled terminal definition is found in /usr/lib/terminfo/a/att4425. (The a is copied from the first letter of att4425 to avoid creation of huge directories.) However, if TERMINFO is set to $HOME/myterms, curses will first check $HOME/myterms/a/att4425, and, if that fails, will then check /usr/lib/terminfo/a/att4425. This is useful for developing experimental definitions or when write permission on /usr/lib/terminfo is not available.

The integer variables LINES and COLS are defined in <curses.h>, and will be filled in by initscr() with the size of the screen. (For more information, see the subsection "Terminfo-Level Manipulations"). The constants TRUE and FALSE have the values 1 and 0, respectively. The constants ERR and OK are returned by routines to indicate whether the routine successfully completed. These constants are also defined in <curses.h>.

**ROUTINES**

Many of the following routines have two or more versions. The routines prefixed with w require a window argument. The routines prefixed with p require a pad argument. Those without a prefix generally use stdscr.

The routines prefixed with mv require y and x coordinates to move to before performing the appropriate action. The mv() routines imply a call to move() before the call to the other routine. The window argument is always specified before the coordinates. y always refers to the row (of the window), and x always refers to the column. The upper left corner is always (0,0), not (1,1). The routines prefixed with mvw take both a window argument and y and x coordinates.

In each case, win is the window affected and pad is the pad affected. (win and pad are always of type WINDOW *) Option-setting routines require a boolean flag bf with the value TRUE or FALSE. (bf is always of type bool) The types WINDOW, bool, and chtype are defined in <curses.h>. See the SYNOPSIS for a summary of what types all variables are.

All routines return either the integer ERR or the integer OK, unless otherwise noted. Routines that return pointers always return NULL on error.

**Overall Screen Manipulation**

**WINDOW *initscr()** The first routine called should almost always be initscr(). (The exceptions are slk_init(), filter(), and ripoffline().) This will determine the terminal type and initialize all curses data structures. initscr() also arranges that the first call to refresh() will clear the screen. If errors occur, initscr() will write an appropriate error message to standard error and exit; otherwise, a pointer to stdscr is returned. If the program wants an indication of error conditions, newterm() should be used instead of initscr(). initscr() should only be called once per application.

**endwin()** A program should always call endwin() before exiting or escaping from curses mode temporarily, to do a shell
escape or system(3S) call, for example. This routine will restore tty(7) modes, move the cursor to the lower left corner of the screen and reset the terminal into the proper non-visual mode. To resume after a temporary escape, call wrefresh() or doupdate().

isendwin() Returns TRUE if endwin() has been called without any subsequent calls to wrefresh().

SCREEN *newterm(type, outfd, infd)
A program that outputs to more than one terminal must use newterm() for each terminal instead of initscr(). A program that wants an indication of error conditions, so that it may continue to run in a line-oriented mode if the terminal cannot support a screen-oriented program, must also use this routine. newterm() should be called once for each terminal. It returns a variable of type SCREEN that should be saved as a reference to that terminal. The arguments are the type of the terminal to be used in place of the environment variable TERM; outfd, a stdio(3S) file pointer for output to the terminal; and infd, another file pointer for input from the terminal. When it is done running, the program must also call endwin() for each terminal being used. If newterm() is called more than once for the same terminal, the first terminal referred to must be the last one for which endwin() is called.

SCREEN *set_term(new)
This routine is used to switch between different terminals. The screen reference new becomes the new current terminal. A pointer to the screen of the previous terminal is returned by the routine. This is the only routine which manipulates SCREEN pointers; all other routines affect only the current terminal.

Window and Pad Manipulation
refresh() wrefresh (win) These routines (or prefresh(), pnoutrefresh(), wnoutrefresh(), or doupdate()) must be called to write output to the terminal, as most other routines merely manipulate data structures. wrefresh() copies the named window to the physical terminal screen, taking into account what is already there in order to minimize the amount of information that's sent to the terminal (called optimization). refresh() does the same thing, except it uses stdscr as a default window. Unless leaveok() has been enabled, the physical cursor of the terminal is left at the location of the window’s cursor. The number of characters output to the terminal is returned.

Note that refresh() is a macro.
These two routines allow multiple updates to the physical terminal screen with more efficiency than `wrefresh()` alone. How this is accomplished is described in the next paragraph.

curses keeps two data structures representing the terminal screen: a physical terminal screen, describing what is actually on the screen, and a virtual terminal screen, describing what the programmer wants to have on the screen. `wrefresh()` works by first calling `wnoutrefresh()`, which copies the named window to the virtual screen, and then by calling `doupdate()`, which compares the virtual screen to the physical screen and does the actual update. If the programmer wishes to output several windows at once, a series of calls to `wrefresh()` will result in alternating calls to `wnoutrefresh()` and `doupdate()`, causing several bursts of output to the screen. By first calling `wnoutrefresh()` for each window, it is then possible to call `doupdate()` once, resulting in only one burst of output, with probably fewer total characters transmitted and certainly less processor time used.

**WINDOW *newwin(nlines, ncols, begin_y, begin_x)**
Create and return a pointer to a new window with the given number of lines (or rows), `nlines`, and columns, `ncols`. The upper left corner of the window is at line `begin_y`, column `begin_x`. If either `nlines` or `ncols` is 0, they will be set to the value of `lines=begin_y` and `cols=begin_x`. A new full-screen window is created by calling `newwin(0,0,0,0).

**mvwin(win, y, x)**
Move the window so that the upper left corner will be at position `(y, x)`. If the move would cause the window to be off the screen, it is an error and the window is not moved.

**WINDOW *subwin(orig, nlines, ncols, begin_y, begin_x)**
Create and return a pointer to a new window with the given number of lines (or rows), `nlines`, and columns, `ncols`. The window is at position `(begin_y, begin_x)` on the screen. (This position is relative to the screen, and not to the window `orig`.) The window is made in the middle of the window `orig`, so that changes made to one window will affect both windows. When using this routine, often it will be necessary to call `touchwin()` or `touchline()` on `orig` before calling `wrefresh()`.

**delwin(win)**
Delete the named window, freeing up all memory associated with it. In the case of overlapping windows, subwindows should be deleted before the main window.
WINDOW *newpad(nlines, ncols)
Create and return a pointer to a new pad data structure with the given number of lines (or rows), nlines, and columns, ncols. A pad is a window that is not restricted by the screen size and is not necessarily associated with a particular part of the screen. Pads can be used when a large window is needed, and only a part of the window will be on the screen at one time. Automatic refreshes of pads (e.g. from scrolling or echoing of input) do not occur. It is not legal to call wrefresh() with a pad as an argument; the routines prefresh() or pnoutrefresh() should be called instead. Note that these routines require additional parameters to specify the part of the pad to be displayed and the location on the screen to be used for display.

WINDOW *subpad(orig, nlines, ncols, begin_y, begin_x)
Create and return a pointer to a subwindow within a pad with the given number of lines (or rows), nlines, and columns, ncols. Unlike subwin(), which uses screen coordinates, the window is at position (begin_y, begin_x) on the pad. The window is made in the middle of the window orig, so that changes made to one window will affect both windows. When using this routine, often it will be necessary to call touchwin() or touchline() on orig before calling prefresh().

prefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)
pnoutrefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)
These routines are analogous to wrefresh() and wnoutrefresh() except that pads, instead of windows, are involved. The additional parameters are needed to indicate what part of the pad and screen are involved. pminrow and pmincol specify the upper left corner, in the pad, of the rectangle to be displayed. sminrow, smincol, smaxrow, smaxcol specify the edges, on the screen, of the rectangle to be displayed in. The lower right corner in the pad of the rectangle to be displayed is calculated from the screen coordinates, since the rectangles must be the same size. Both rectangles must be entirely contained within their respective structures. Negative values of pminrow, pmincol, sminrow, or smincol are treated as if they were zero.
Output

These routines are used to "draw" text on windows.

addch(ch)

waddch(win, ch)

mvaddch(y, x, ch)

mvwaddch(win, y, x, ch)

The character ch is put into the window at the current cursor position of the window and the position of the window cursor is advanced. Its function is similar to that of putchar (see putc(3S)). At the right margin, an automatic newline is performed. At the bottom of the scrolling region, if scrollok() is enabled, the scrolling region will be scrolled up one line.

If ch is a tab, newline, or backspace, the cursor will be moved appropriately within the window. A newline also does a clrtoeol() before moving. Tabs are considered to be at every eighth column. If ch is another control character, it will be drawn in the ^X notation. (Calling winch() after adding a control character will not return the control character, but instead will return the representation of the control character.)

Video attributes can be combined with a character by or-ing them into the parameter. This will result in these attributes also being set. (The intent here is that text, including attributes, can be copied from one place to another using inch() and addch(),) See standout(), below.

Note that ch is actually of type chtype, not a character.

Note that addch(), mvaddch(), and mvwaddch(), are macros.

echochar(ch)

ewchochar(win, ch)

echochar(pad, ch)

These routines are functionally equivalent to a call to addch(ch) followed by a call to refresh(), a call to waddch(win, ch) followed by a call to wrefresh(win), or a call to waddch(pad, ch) followed by a call to refresh(pad). The knowledge that only a single character is being output is taken into consideration and, for non-control characters, a considerable performance gain can be seen by using these routines instead of their equivalents. In the case of echochar(), the last location of the pad on the screen is reused for the arguments to refresh().

Note that ch is actually of type chtype, not a character.

Note that echochar() is a macro.
addstr(str)
waddstr(win, str)
mvwaddstr(win, y, x, str)
mvaddstr(y, x, str)

These routines write all the characters of the null-terminated character string str on the given window. This is equivalent to calling waddch() once for each character in the string.

Note that addstr(), mvaddstr(), and mvwaddstr() are macros.

attroff(attrs)
wattroff(win, attrs)
attron(attrs)
wattron(win, attrs)
attrset(attrs)
wattrset(win, attrs)
standend()
wstandend(win)
standout()
wstandout(win)

These routines manipulate the current attributes of the named window. These attributes can be any combination of A_STANDOUT, A_REVERSE, A_BOLD, A_DIM, A_BLINK, A_UNDERLINE, and A_ALTCHARSET. These constants are defined in <curses.h> and can be combined with the C logical OR ( | ) operator.

The current attributes of a window are applied to all characters that are written into the window with waddch(). Attributes are a property of the character, and move with the character through any scrolling and insert/delete line/character operations. To the extent possible on the particular terminal, they will be displayed as the graphic rendition of the characters put on the screen.

attrset(attrs) sets the current attributes of the given window to attrs. attroff(attrs) turns off the named attributes without turning on or off any other attributes. attron(attrs) turns on the named attributes without affecting any others. standout() is the same as attron(A_STANDOUT). standend() is the same as attrset (0), that is, it turns off all attributes.

Note that attrs is actually of type chtype, not a character.

Note that attroff(), attron(), attrset(), standend(), and standout() are macros.

beep() flash()

These routines are used to signal the terminal user. beep() will sound the audible alarm on the terminal, if possible, and if not, will flash the screen (visible bell), if that is possible. flash() will flash the screen, and if that is
not possible, will sound the audible signal. If neither signal is possible, nothing will happen. Nearly all terminals have an audible signal (bell or beep) but only some can flash the screen.

**box**(win, vertch, horch)

A box is drawn around the edge of the window, win. vertch and horch are the characters the box is to be drawn with. If vertch and horch are 0, then appropriate default characters, ACS_VLINE and ACS_HLINE, will be used.

Note that vertch and horch are actually of type chtype, not characters.

**erase()**

**werase**(win)

These routines copy blanks to every position in the window.

Note that **erase()** is a macro.

**clear()**

**wclear**(win)

These routines are like **erase()** and **werase()**, but they also call **clearok()**, arranging that the screen will be cleared completely on the next call to **wrefresh()** for that window, and repainted from scratch.

Note that **clear()** is a macro.

**clrtobot()**

**wclrtobot**(win)

All lines below the cursor in this window are erased. Also, the current line to the right of the cursor, inclusive, is erased.

Note that **clrtobot()** is a macro.

**clrtoeol()**

**wclrtoeol**(win)

The current line to the right of the cursor, inclusive, is erased.

Note that **clrtoeol()** is a macro.

**delay_output**(ms)

Insert a ms millisecond pause in the output. It is not recommended that this routine be used extensively, because padding characters are used rather than a processor pause.

**delch()**

**wdelch**(win)

**mvdelch**(y, x)

**mvwdelch**(win, y, x)

The character under the cursor in the window is deleted. All characters to the right on the same line are moved to the left one position and the last character on the line is filled with a blank. The cursor position does not change (after moving to (y, x), if specified). (This does not imply use of the hardware "delete-character" feature.)
CURSES(3X)  (Terminal Information Utilities)  CURSES(3X)

Note that delch(), mvdelch(), and mvwdelch() are macros.

**deleteln()**

The line under the cursor in the window is deleted. All lines below the current line are moved up one line. The bottom line of the window is cleared. The cursor position does not change. (This does not imply use of the hardware "delete-line" feature.)

Note that deleteln() is a macro.

**getyx(win, y, x)**

The cursor position of the window is placed in the two integer variables \( y \) and \( x \). This is implemented as a macro, so no "&" is necessary before the variables.

**getbegyx(win, y, x)**

**getmaxyx(win, y, x)**

Like getyx(), these routines store the current beginning coordinates and size of the specified window.

Note that getbegyx() and getmaxyx() are macros.

**insch(ch)**

The character \( ch \) is inserted before the character under the cursor. All characters to the right are moved one space to the right, possibly losing the rightmost character of the line. The cursor position does not change (after moving to \((y, x)\), if specified). (This does not imply use of the hardware "insert-character" feature.)

Note that \( ch \) is actually of type chtype, not a character.

Note that insch(), mvinsch(), and mvwinsch() are macros.

**insertln()**

A blank line is inserted above the current line and the bottom line is lost. (This does not imply use of the hardware "insert-line" feature.)

Note that insertln() is a macro.

**move(y, x)**

The cursor associated with the window is moved to line (row) \( y \), column \( x \). This does not move the physical cursor of the terminal until refresh() is called. The position specified is relative to the upper left corner of the window, which is \((0, 0)\).

Note that move() is a macro.

**overlay(srcwin, dstwin)**
overwrite(srcwin, dstwin)

These routines overlay srcwin on top of dstwin; that is, all text in srcwin is copied into dstwin. srcwin and dstwin need not be the same size; only text where the two windows overlap is copied. The difference is that overlay() is non-destructive (blanks are not copied), while overwrite() is destructive.

copywin(srcwin, dstwin, sminrow, smincol, dminrow, dmincol, dmaxrow, dmaxcol, overlay)

This routine provides a finer grain of control over the overlay() and overwrite() routines. Like in the prefresh() routine, a rectangle is specified in the destination window, (dminrow, dmincol) and (dmaxrow, dmaxcol), and the upper-left-corner coordinates of the source window, (sminrow, smincol). If the argument overlay is true, then copying is non-destructive, as in overlay().

printw(fmt [, arg...])
wprintw(win, fmt [, arg...])

mvprintw(y, x, fmt [, arg...])
mvwprintw(win, y, x, fmt [, arg...])

These routines are analogous to printf(3). The string which would be output by printf(3) is instead output using waddstr() on the given window.

vwprintw(win, fmt, varglist)

This routine corresponds to vfprintf(3S). It performs a wprintw() using a variable argument list. The third argument is a va_list, a pointer to a list of arguments, as defined in <varargs.h>. See the vprintf(3S) and varargs(5) manual pages for a detailed description on how to use variable argument lists.

scroll(win)

The window is scrolled up one line. This involves moving the lines in the window data structure. As an optimization, if the window is stdscr and the scrolling region is the entire window, the physical screen will be scrolled at the same time.

touchwin(win)
touchline(win, start, count)

Throw away all optimization information about which parts of the window have been touched, by pretending that the entire window has been drawn on. This is sometimes necessary when using overlapping windows, since a change to one window will affect the other window, but the records of which lines have been changed in the other window will not reflect the change. touchline() only pretends that count lines have been changed, beginning with line start.
Input

getch()
wgetch(win)
mvgetch(y, x)
mvwgetch(win, y, x) A character is read from the terminal associated with the window. In NODELAY mode, if there is no input waiting, the value ERR is returned. In DELAY mode, the program will hang until the system passes text through to the program. Depending on the setting of cbreak(), this will be after one character (CBREAK mode), or after the first newline (NOCBREAK mode). In HALF-DELAY mode, the program will hang until a character is typed or the specified timeout has been reached. Unless noecho() has been set, the character will also be echoed into the designated window. No refresh() will occur between the move() and the getch() done within the routines mvgetch() and mvwgetch().

When using getch(), wgetch(), mvgetch(), or mvwgetch(), do not set both NOCBREAK mode (nocbreak()) and ECHO mode (echo()) at the same time. Depending on the state of the tty(7) driver when each character is typed, the program may produce undesirable results.

If keypad(win, TRUE) has been called, and a function key is pressed, the token for that function key will be returned instead of the raw characters. (See keypad() under "Input Options Setting.") Possible function keys are defined in <curses.h> with integers beginning with 0401, whose names begin with KEY_. If a character is received that could be the beginning of a function key (such as escape), curses will set a timer. If the remainder of the sequence is not received within the designated time, the character will be passed through, otherwise the function key value will be returned. For this reason, on many terminals, there will be a delay after a user presses the escape key before the escape is returned to the program. (Use by a programmer of the escape key for a single character routine is discouraged. Also see notimeout() below.)

Note that getch(), mvgetch(), and mvwgetch() are macros.

gestr(str)
wgestr(win, str)
mvgestr(y, x, str)
mvwgestr(win, y, x, str) A series of calls to getch() is made, until a newline, carriage return, or enter key is received. The resulting value is placed in the area pointed at by the character pointer.
CURSES(3X) (Terminal Information Utilities) CURSES(3X)

str. The user’s erase and kill characters are interpreted. As in mvgetch(), no refresh() is done between the move() and getstr() within the routines mvgetstr() and mvwgetstr().

Note that getstr(), mvgetstr(), and mvwgetstr() are macros.

flushinp() Throws away any typeahead that has been typed by the user and has not yet been read by the program.

ungetch(c) Place c back onto the input queue to be returned by the next call to wgetch().

inch() winch(win) mvinch(y, x) mvwinch(win, y, x) The character, of type chtype, at the current position in the named window is returned. If any attributes are set for that position, their values will be OR’ed into the value returned. The predefined constants A_CHARTEXT and A_ATTRIBUTES, defined in <curses.h>, can be used with the C logical AND (&) operator to extract the character or attributes alone.

Note that inch(), winch(), mvinch(), and mvwinch() are macros.

scanw(fmt [, arg ... J]) wscanw(win, fmt [, arg...] mvscanw(y, x, fmt [, arg...] mvwscanw(win, y, x, fmt [, arg...] These routines correspond to scanf(3S), as do their arguments and return values. wgetstr() is called on the window, and the resulting line is used as input for the scan.

vwscanw(win, fmt, ap) This routine is similar to vwprintw() above in that performs a wscanw() using a variable argument list. The third argument is a va_list, a pointer to a list of arguments, as defined in <varargs.h>. See the vprintf(3S) and varargs(5) manual pages for a detailed description on how to use variable argument lists.

Output Options Setting These routines set options within curses that deal with output. All options are initially FALSE, unless otherwise stated. It is not necessary to turn these options off before calling endwin().

clearok(win, bf) If enabled (bf is TRUE), the next call to wrefresh() with this window will clear the screen completely and redraw the entire screen from scratch. This is useful when the
contents of the screen are uncertain, or in some cases for a more pleasing visual effect.

**idlok(win, bf)**

If enabled (bf is TRUE), curses will consider using the hardware "insert/delete-line" feature of terminals so equipped. If disabled (bf is FALSE), curses will very seldom use this feature. (The "insert/delete-character" feature is always considered.) This option should be enabled only if your application needs "insert/delete-line", for example, for a screen editor. It is disabled by default because "insert/delete-line" tends to be visually annoying when used in applications where it isn't really needed. If "insert/delete-line" cannot be used, curses will redraw the changed portions of all lines.

**leaveok(win, bf)**

Normally, the hardware cursor is left at the location of the window cursor being refreshed. This option allows the cursor to be left wherever the update happens to leave it. It is useful for applications where the cursor is not used, since it reduces the need for cursor motions. If possible, the cursor is made invisible when this option is enabled.

**setscrreg(top, bot)**

**wsetscrreg(win, top, bot)**

These routines allow the user to set a software scrolling region in a window. top and bot are the line numbers of the top and bottom margin of the scrolling region. (Line 0 is the top line of the window.) If this option and **scrollok()** are enabled, an attempt to move off the bottom margin line will cause all lines in the scrolling region to scroll up one line. (Note that this has nothing to do with use of a physical scrolling region capability in the terminal, like that in the DEC VT100. Only the text of the window is scrolled; if **idlok()** is enabled and the terminal has either a scrolling region or "insert/delete-line" capability, they will probably be used by the output routines.)

Note that **setscrreg()** and **wsetscrreg()** are macros.

**scrollok(win, bf)**

This option controls what happens when the cursor of a window is moved off the edge of the window or scrolling region, either from a newline on the bottom line, or typing the last character of the last line. If disabled (bf is FALSE), the cursor is left on the bottom line at the location where the offending character was entered. If enabled (bf is TRUE), **wrefresh()** is called on the window, and then the physical terminal and window are scrolled up one line. (Note that in order to get the physical scrolling effect on the terminal, it is also necessary to call **idlok()**.)
nl()
nonl()

These routines control whether newline is translated into carriage return and linefeed on output, and whether return is translated into newline on input. Initially, the translations do occur. By disabling these translations using nonl(), curses is able to make better use of the linefeed capability, resulting in faster cursor motion.

Input Options Setting
These routines set options within curses that deal with input. The options involve using ioctl(2) and therefore interact with curses routines. It is not necessary to turn these options off before calling endwin().

For more information on these options, see Chapter 10 of the Programmer’s Guide.

cbreak()
nocbreak()

These two routines put the terminal into and out of CBREAK mode, respectively. In CBREAK mode, characters typed by the user are immediately available to the program and erase/kill character processing is not performed. When in NOCBREAK mode, the tty driver will buffer characters typed until a newline or carriage return is typed. Interrupt and flow-control characters are unaffected by this mode (see termio(7)). Initially the terminal may or may not be in CBREAK mode, as it is inherited, therefore, a program should call cbreak() or nocbreak() explicitly. Most interactive programs using curses will set CBREAK mode.

Note that cbreak() overrides raw(). See getch() under "Input" for a discussion of how these routines interact with echo() and noecho().

echo()
noecho()

These routines control whether characters typed by the user are echoed by getch() as they are typed. Echoing by the tty driver is always disabled, but initially getch() is in ECHO mode, so characters typed are echoed. Authors of most interactive programs prefer to do their own echoing in a controlled area of the screen, or not to echo at all, so they disable echoing by calling noecho(). See getch() under "Input" for a discussion of how these routines interact with cbreak() and nocbreak().

halfdelay(tenths)

Half-delay mode is similar to CBREAK mode in that characters typed by the user are immediately available to the program. However, after blocking for tenths tenths of seconds, ERR will be returned if nothing has been typed. tenths must be a number between 1 and 255. Use nocbreak() to leave half-delay mode.
intrflush(win, bf) If this option is enabled, when an interrupt key is pressed on the keyboard (interrupt, break, quit) all output in the tty driver queue will be flushed, giving the effect of faster response to the interrupt, but causing curses to have the wrong idea of what is on the screen. Disabling the option prevents the flush. The default for the option is inherited from the tty driver settings. The window argument is ignored.

keypad(win, bf) This option enables the keypad of the user’s terminal. If enabled, the user can press a function key (such as an arrow key) and wgetch() will return a single value representing the function key, as in KEY_LEFT. If disabled, curses will not treat function keys specially and the program would have to interpret the escape sequences itself. If the keypad in the terminal can be turned on (made to transmit) and off (made to work locally), turning on this option will cause the terminal keypad to be turned on when wgetch() is called.

meta(win, bf) If enabled, characters returned by wgetch() are transmitted with all 8 bits, instead of with the highest bit stripped. In order for meta() to work correctly, the km (has_meta_key) capability has to be specified in the terminal’s terminfo(4) entry.

nodelay(win, bf) This option causes wgetch() to be a non-blocking call. If no input is ready, wgetch() will return ERR. If disabled, wgetch() will hang until a key is pressed.

notimeout(win, bf) While interpreting an input escape sequence, wgetch() will set a timer while waiting for the next character. If notimeout(win, TRUE) is called, then wgetch() will not set a timer. The purpose of the timeout is to differentiate between sequences received from a function key and those typed by a user.

raw() noraw() The terminal is placed into or out of raw mode. RAW mode is similar to CBREAK mode, in that characters typed are immediately passed through to the user program. The differences are that in RAW mode, the interrupt, quit, suspend, and flow control characters are passed through uninterpreted, instead of generating a signal. RAW mode also causes 8-bit input and output. The behavior of the BREAK key depends on other bits in the tty(7) driver that are not set by curses.

typeahead(fildes) curses does “line-breakout optimization” by looking for typeahead periodically while updating the screen. If input is found, and it is coming from a tty, the current update will be postponed until refresh() or douupdate() is called again. This allows faster response to commands typed in
cururses(3x) (terminal information utilities) cururses(3x)

advance. Normally, the file descriptor for the input file
pointer passed to newterm(), or stdin in the case that
initscr() was used, will be used to do this typeahead
checking. The typeahead() routine specifies that the file
descriptor fildes is to be used to check for typeahead
instead. If fildes is −1, then no typeahead checking will be
done.

Note that fildes is a file descriptor, not a <stdio.h> FILE
pointer.

Environment Queries

baudrate() Returns the output speed of the terminal. The number
returned is in bits per second, for example, 9600, and is an
integer.

cchar erasechar() The user's current erase character is returned.

has ic() True if the terminal has insert- and delete-character ca­
apabilities.

has il() True if the terminal has insert- and delete-line capabilities,
or can simulate them using scrolling regions. This might
be used to check to see if it would be appropriate to turn
on physical scrolling using scrollok().

char killchar() The user's current line-kill character is returned.

char *longname() This routine returns a pointer to a static area containing a
verbose description of the current terminal. The max­
imum length of a verbose description is 128 characters. It
is defined only after the call to initscr() or newterm().
The area is overwritten by each call to newterm() and is
not restored by set_term(), so the value should be saved
between calls to newterm() if longname() is going to be
used with multiple terminals.

Soft Labels

If desired, curses will manipulate the set of soft function-key labels that exist on
many terminals. For those terminals that do not have soft labels, if you want to
simulate them, curses will take over the bottom line of stdscr, reducing the size
of stdscr and the variable LINES. curses standardizes on 8 labels of 8 characters
each.

slk init(labfmt) In order to use soft labels, this routine must be called
before initscr() or newterm() is called. If initscr() winds
up using a line from stdscr to emulate the soft labels, then
labfmt determines how the labels are arranged on the
screen. Setting labfmt to 0 indicates that the labels are to
be arranged in a 3-2-3 arrangement; 1 asks for a 4-4
arrangement.

slk set(labnum, label, labfmt)

labnum is the label number, from 1 to 8. label is the string
to be put on the label, up to 8 characters in length. A
CURSES(3X) (Terminal Information Utilities) CURSES(3X)

NULL string or a NULL pointer will put up a blank label. 

labfmt is one of 0, 1 or 2, to indicate whether the label is
to be left-justified, centered, or right-justified within the
label.

slk_refresh()

slk_noutrefresh() These routines correspond to the routines wrefresh() and
wnoutrefresh(). Most applications would use
slk_noutrefresh() because a wrefresh() will most likely
soon follow.

char *slk_label(labnum)

The current label for label number labnum, with leading
and trailing blanks stripped, is returned.

slk_clear() The soft labels are cleared from the screen.

slk_restore() The soft labels are restored to the screen after a
slk_clear().

slk_touch() All of the soft labels are forced to be output the next time
a slk_noutrefresh() is performed.

Low-Level curses Access

The following routines give low-level access to various curses functionality.
These routines typically would be used inside of library routines.

def_prog_mode()
def_shell_mode() Save the current terminal modes as the “program” (in
curses) or “shell” (not in curses) state for use by the
reset_prog_mode() and reset_shell_mode() routines. This
is done automatically by initscr().

reset_prog_mode() Restore the terminal to “program” (in curses) or “shell”
(out of curses) state. These are done automatically by
endwin() and doupdate() after an endwin(), so they nor-
mally would not be called.

resetty() savetty() These routines save and restore the state of the terminal
modes. savetty() saves the current state of the terminal in
a buffer and resetty() restores the state to what it was at
the last call to savetty().

getsyx(y, x) The current coordinates of the virtual screen cursor are
returned in y and x. Like getyx(), the variables y and x do
not take an “&” before them. If leaveok() is currently
TRUE, then −1, −1 will be returned. If lines may have
been removed from the top of the screen using ripoffline() and the values are to be used beyond just
passing them on to setsyx(), the value
y+stdscr->yoffset should be used for those other uses.
Note that getsyx() is a macro.
setsyx(y, x)
The virtual screen cursor is set to y, x. If y and x are both
-1, then leaveok() will be set. The two routines setsyx() and
setsyx() are designed to be used by a library routine
which manipulates curses windows but does not want to
mess up the current position of the program’s cursor. The
library routine would call getsyx() at the beginning, do its
manipulation of its own windows, do a wnoutrefresh() on
its windows, call setsyx(), and then call doupdate().

ripoffline(line, init)
This routine provides access to the same facility that
slk_init() uses to reduce the size of the screen.
ripoffline() must be called before initscr() or newterm() is
called. If line is positive, a line will be removed from
the top of stdscr; if negative, a line will be removed from
the bottom. When this is done inside initscr(), the rou­
tine init() is called with two arguments: a window pointer
to the 1-line window that has been allocated and an
integer with the number of columns in the window. Inside this initialization routine, the integer variables
LINES and COLS (defined in <curses.h>) are not
guaranteed to be accurate and wrefresh() or doupdate() must not be called. It is allowable to call wnoutrefresh() during the initialization routine.

ripoffline() can be called up to five times before calling
initscr() or newterm().

scr_dump(filename) The current contents of the virtual screen are written to
the file filename.

scr_restore(filename) The virtual screen is set to the contents of filename, which
must have been written using scr_dump(). The next call
to doupdate() will restore the screen to what it looked
like in the dump file.

scr_init(filename) The contents of filename are read in and used to initialize
the curses data structures about what the terminal
currently has on its screen. If the data is determined to be
valid, curses will base its next update of the screen on this
information rather than clearing the screen and starting
from scratch. scr_init() would be used after initscr() or a
system(3S) call to share the screen with another process
which has done a scr_dump() after its endwin() call. The
data will be declared invalid if the time-stamp of the tty is
old or the terminfo(4) capability nrmc is true.

curs_set(visibility) The cursor is set to invisible, normal, or very visible for
visibility equal to 0, 1 or 2.

draino(ms) Wait until the output has drained enough that it will only
take ms more milliseconds to drain completely.
garbagedlines(win, begline, numlines)
This routine indicates to curses that a screen line is garbaged and should be thrown away before having anything written over the top of it. It could be used for programs such as editors which want a command to redraw just a single line. Such a command could be used in cases where there is a noisy communications line and redrawing the entire screen would be subject to even more communication noise. Just redrawing the single line gives some semblance of hope that it would show up unblemished. The current location of the window is used to determine which lines are to be redrawn.

napms(ms)
Sleep for ms milliseconds.

**Termino-Level Manipulations**
These low-level routines must be called by programs that need to deal directly with the terminfo(4) database to handle certain terminal capabilities, such as programming function keys. For all other functionality, curses routines are more suitable and their use is recommended.

Initially, setupterm() should be called. (Note that setupterm() is automatically called by initscr() and newterm().) This will define the set of terminal-dependent variables defined in the terminfo(4) database. The terminfo(4) variables lines and columns (see terminfo(4)) are initialized by setupterm() as follows: if the environment variables LINES and COLUMNS exist, their values are used. If the above environment variables do not exist and the program is running in a layer (see layers(1)), the size of the current layer is used. Otherwise, the values for lines and columns specified in the terminfo(4) database are used.

The header files <curses.h> and <term.h> should be included, in this order, to get the definitions for these strings, numbers, and flags. Parameterized strings should be passed through tparm() to instantiate them. All terminfo(4) strings (including the output of tparm()) should be printed with tputs() or putp(). Before exiting, reset_shell_mode() should be called to restore the tty modes. Programs which use cursor addressing should output enter_ca_mode upon startup and should output exit_ca_mode before exiting (see terminfo(4)). Programs desiring shell escapes should call reset_shell_mode() and output exit_ca_mode before the shell is called and should output enter_ca_mode and call reset_prog_mode() after returning from the shell. Note that this is different from the curses routines (see endwin()).

setupterm(term, fildes, errret)
Reads in the terminfo(4) database, initializing the terminfo(4) structures, but does not set up the output virtualization structures used by curses. The terminal type is in the character string term; if term is NULL, the environment variable TERM will be used. All output is to the file descriptor fildes. If errret is not NULL, then setupterm() will return OK or ERR and store a status value in the integer pointed to by errret. A status of 1 in errret is normal, 0 means that the terminal could not be found, and
-1 means that the terminfo(4) database could not be found. If errret is NULL, setupterm() will print an error message upon finding an error and exit. Thus, the simplest call is setupterm ((char *)0, 1, (int *)0), which uses all the defaults.

The terminfo(4) boolean, numeric and string variables are stored in a structure of type TERMINAL. After setupterm() returns successfully, the variable cur_term (of type TERMINAL *) is initialized with all of the information that the terminfo(4) boolean, numeric and string variables refer to. The pointer may be saved before calling setupterm() again. Further calls to setupterm() will allocate new space rather than reuse the space pointed to by cur_term.

set_curterm(nterm) nterm is of type TERMINAL *. set_curterm() sets the variable cur_term to nterm, and makes all of the terminfo(4) boolean, numeric and string variables use the values from nterm.

del_curterm(oterm) oterm is of type TERMINAL *. del_curterm() frees the space pointed to by oterm and makes it available for further use. If oterm is the same as cur_term, then references to any of the term info(4) boolean, numeric and string variables thereafter may refer to invalid memory locations until another setupterm() has been called.

restartterm(term, fildes, errret) Like setupterm() after a memory restore.

char *tparm(str, ...)

Instantiate the string str with parms p.. A pointer is returned to the result of str with the parameters applied.

tputs(str, count, putc)

Apply padding to the string str and output it. str must be a terminfo(4) string variable or the return value from tparm(), tgetstr(), tigetstr() or tgoto(). count is the number of lines affected, or 1 if not applicable. putc() is a putchar(3S)-like routine to which the characters are passed, one at a time.

putp(str) A routine that calls tputs (str, 1, putchar()).

vidputs(attrs, putc) Output a string that puts the terminal in the video attribute mode attrs, which is any combination of the attributes listed below. The characters are passed to the putchar(3S)-like routine putc().

vidattr(attrs) Like vidputs(), except that it outputs through putchar(3S).

mvcur(oldrow, oldcol, newrow, newcol)

Low-level cursor motion.
The following routines return the value of the capability corresponding to the `terminfo(4)` `capname` passed to them, such as `xenl`.

- `tigetflag(capname)` The value $-1$ is returned if `capname` is not a boolean capability.
- `tigetnum(capname)` The value $-2$ is returned if `capname` is not a numeric capability.
- `tigetstr(capname)` The value `(char *)$-1$ is returned if `capname` is not a string capability.

`char *boolnames[]`, `*boolcodes[]`, `*boolfnames[]`  
`char *numnames[]`, `*numcodes[]`, `*numfnames[]`  
`char *strnames[]`, `*strcodes[]`, `*strfnames[]`  

These null-terminated arrays contain the `capnames`, the `termcap` codes, and the full C names, for each of the `terminfo(4)` variables.

**Termcap Emulation**

These routines are included as a conversion aid for programs that use the `termcap` library. Their parameters are the same and the routines are emulated using the `terminfo(4)` database.

- `tgetent(bp, name)` Look up `termcap` entry for `name`. The emulation ignores the buffer pointer `bp`.
- `tgetflag(codename)` Get the boolean entry for `codename`.
- `tgetnum(codes)` Get numeric entry for `codename`.
- `char *tgetstr(codename, area)` Return the string entry for `codename`. If `area` is not `NULL`, then also store it in the buffer pointed to by `area` and advance `area`. `tputs()` should be used to output the returned string.
- `char *tgoto(cap, col, row)` Instantiate the parameters into the given capability. The output from this routine is to be passed to `tputs()`.
- `tputs(str, affcnt, putc)` See `tputs()` above, under "Termino-Level Manipulations".

**Miscellaneous**

- `traceoff()`  
- `traceon()`  
- `unctrl(c)`  

`traceoff()` and `traceon()` turn off and on debugging trace output when using the debug version of the `curses` library, `/usr/lib/libcurses.a`. This facility is available only to customers with a source license.

This macro expands to a character string which is a printable representation of the character `c`. Control characters are displayed in the `\'X notation. Printing characters are displayed as is.
unctrl() is a macro, defined in `<unctrl.h>`, which is automatically included by `<curses.h>`.

```c
char *keyname(c)  
```
A character string corresponding to the key c is returned.

```c
filter()          
```
This routine is one of the few that is to be called before `initscr()` or `newterm()` is called. It arranges things so that `curses` thinks that there is a 1-line screen. `curses` will not use any terminal capabilities that assume that they know what line on the screen the cursor is on.

Use of curscr

The special window `curscr` can be used in only a few routines. If the window argument to `clearok()` is `curscr`, the next call to `wrefresh()` with any window will cause the screen to be cleared and repainted from scratch. If the window argument to `wrefresh()` is `curscr`, the screen is immediately cleared and repainted from scratch. (This is how most programs would implement a "repaint-screen" routine.) The source window argument to `overlay()`, `overwrite()`, and `copywin()` may be `curscr`, in which case the current contents of the virtual terminal screen will be accessed.

Obsoleted Calls

Various routines are provided to maintain compatibility in programs written for older versions of the `curses` library. These routines are all emulated as indicated below.

```c
crmode()          
```
Replaced by `cbreak()`.

```c
fixterm()         
```
Replaced by `reset_prog_mode()`.

```c
gettmode()        
```
A no-op.

```c
nocrmode()        
```
Replaced by `nocbreak()`.

```c
resetterm()       
```
Replaced by `reset_shell_mode()`.

```c
saveterm()        
```
Replaced by `def_shell_mode()`.

```c
setterm()         
```
Replaced by `setupterm()`.

**ATTRIBUTES**

The following video attributes, defined in `<curses.h>`, can be passed to the routines `attron()`, `attroff()`, and `attrset()`, or OR'ed with the characters passed to `addch()`.

- **A_STANDOUT** Terminal's best highlighting mode
- **A_UNDERLINE** Underlining
- **A_REVERSE** Reverse video
- **A_BLINK** Blinking
- **A_DIM** Half bright
- **A_BOLD** Extra bright or bold
- **A_ALTCCHARSET** Alternate character set
- **A_CHARTEXT** Bit-mask to extract character (described under `winch()`)

---

444
A_ATTRIBUTES Bit-mask to extract attributes (described under winch())
A_NORMAL Bit mask to reset all attributes off
(for example: attrset (A_NORMAL)

FUNCTION-KEYS

The following function keys, defined in <curses.h>, might be returned by
getch() if keypad() has been enabled. Note that not all of these may be sup­
ported on a particular terminal if the terminal does not transmit a unique code
when the key is pressed or the definition for the key is not present in the termi­
minofo(4) database.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Key name</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_BREAK</td>
<td>0401</td>
<td>break key (unreliable)</td>
</tr>
<tr>
<td>KEY_DOWN</td>
<td>0402</td>
<td>The four arrow keys ...</td>
</tr>
<tr>
<td>KEY_UP</td>
<td>0403</td>
<td></td>
</tr>
<tr>
<td>KEY_LEFT</td>
<td>0404</td>
<td></td>
</tr>
<tr>
<td>KEY_RIGHT</td>
<td>0405</td>
<td></td>
</tr>
<tr>
<td>KEY_HOME</td>
<td>0406</td>
<td>Home key (upward+left arrow)</td>
</tr>
<tr>
<td>KEY_BACKSPACE</td>
<td>0407</td>
<td>backspace (unreliable)</td>
</tr>
<tr>
<td>KEY_F0</td>
<td>0410</td>
<td>Function keys. Space for 64 keys is reserved.</td>
</tr>
<tr>
<td>KEY_F(n)</td>
<td>(KEY_F0+(n))</td>
<td>Formula for $f_n$.</td>
</tr>
<tr>
<td>KEY_DL</td>
<td>0510</td>
<td>Delete line</td>
</tr>
<tr>
<td>KEY_IL</td>
<td>0511</td>
<td>Insert line</td>
</tr>
<tr>
<td>KEY_DC</td>
<td>0512</td>
<td>Delete character</td>
</tr>
<tr>
<td>KEY_IC</td>
<td>0513</td>
<td>Insert char or enter insert mode</td>
</tr>
<tr>
<td>KEY_EIC</td>
<td>0514</td>
<td>Exit insert char mode</td>
</tr>
<tr>
<td>KEY_CLEAR</td>
<td>0515</td>
<td>Clear screen</td>
</tr>
<tr>
<td>KEY_EOS</td>
<td>0516</td>
<td>Clear to end of screen</td>
</tr>
<tr>
<td>KEY_EOL</td>
<td>0517</td>
<td>Clear to end of line</td>
</tr>
<tr>
<td>KEY_SF</td>
<td>0520</td>
<td>Scroll 1 line forward</td>
</tr>
<tr>
<td>KEY_SR</td>
<td>0521</td>
<td>Scroll 1 line backwards (reverse)</td>
</tr>
<tr>
<td>KEY_NPAGE</td>
<td>0522</td>
<td>Next page</td>
</tr>
<tr>
<td>KEY_PPAGE</td>
<td>0523</td>
<td>Previous page</td>
</tr>
<tr>
<td>KEY_STAB</td>
<td>0524</td>
<td>Set tab</td>
</tr>
<tr>
<td>KEY_CTAB</td>
<td>0525</td>
<td>Clear tab</td>
</tr>
<tr>
<td>KEY_CATAB</td>
<td>0526</td>
<td>Clear all tabs</td>
</tr>
<tr>
<td>KEY_ENTER</td>
<td>0527</td>
<td>Enter or send</td>
</tr>
<tr>
<td>KEY_SRESET</td>
<td>0530</td>
<td>soft (partial) reset</td>
</tr>
<tr>
<td>KEY_RESET</td>
<td>0531</td>
<td>reset or hard reset</td>
</tr>
<tr>
<td>KEY_PRINT</td>
<td>0532</td>
<td>print or copy</td>
</tr>
<tr>
<td>KEY_LL</td>
<td>0533</td>
<td>home down or bottom (lower left)</td>
</tr>
</tbody>
</table>

keypad is arranged like this:

A1 up A3
left B2 right
C1 down C3

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Key name</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_A1</td>
<td>0534</td>
<td>Upper left of keypad</td>
</tr>
<tr>
<td>KEY_A3</td>
<td>0535</td>
<td>Upper right of keypad</td>
</tr>
<tr>
<td>KEY_B2</td>
<td>0536</td>
<td>Center of keypad</td>
</tr>
<tr>
<td>KEY_C1</td>
<td>0537</td>
<td>Lower left of keypad</td>
</tr>
<tr>
<td>Key</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>KEY_C3</td>
<td>0540</td>
<td>Lower right of keypad</td>
</tr>
<tr>
<td>KEY_BTAB</td>
<td>0541</td>
<td>Back tab key</td>
</tr>
<tr>
<td>KEY_BEG</td>
<td>0542</td>
<td>begin (inning) key</td>
</tr>
<tr>
<td>KEY_CANCEL</td>
<td>0543</td>
<td>cancel key</td>
</tr>
<tr>
<td>KEY_CLOSE</td>
<td>0544</td>
<td>close key</td>
</tr>
<tr>
<td>KEY_COMMAND</td>
<td>0545</td>
<td>cmd (command) key</td>
</tr>
<tr>
<td>KEY_COPY</td>
<td>0546</td>
<td>copy key</td>
</tr>
<tr>
<td>KEY_CREATE</td>
<td>0547</td>
<td>create key</td>
</tr>
<tr>
<td>KEY_END</td>
<td>0550</td>
<td>end key</td>
</tr>
<tr>
<td>KEY_EXIT</td>
<td>0551</td>
<td>exit key</td>
</tr>
<tr>
<td>KEY_FIND</td>
<td>0552</td>
<td>find key</td>
</tr>
<tr>
<td>KEY_HELP</td>
<td>0553</td>
<td>help key</td>
</tr>
<tr>
<td>KEY_MARK</td>
<td>0554</td>
<td>mark key</td>
</tr>
<tr>
<td>KEY_MESSAGE</td>
<td>0555</td>
<td>message key</td>
</tr>
<tr>
<td>KEY_MOVE</td>
<td>0556</td>
<td>move key</td>
</tr>
<tr>
<td>KEY_NEXT</td>
<td>0557</td>
<td>next object key</td>
</tr>
<tr>
<td>KEY_OPEN</td>
<td>0560</td>
<td>open key</td>
</tr>
<tr>
<td>KEY_OPTIONS</td>
<td>0561</td>
<td>options key</td>
</tr>
<tr>
<td>KEY_PREVIOUS</td>
<td>0562</td>
<td>previous object key</td>
</tr>
<tr>
<td>KEY_REDO</td>
<td>0563</td>
<td>redo key</td>
</tr>
<tr>
<td>KEY_REFERENCE</td>
<td>0564</td>
<td>reference key</td>
</tr>
<tr>
<td>KEY_REFRESH</td>
<td>0565</td>
<td>refresh key</td>
</tr>
<tr>
<td>KEY_REPLACE</td>
<td>0566</td>
<td>replace key</td>
</tr>
<tr>
<td>KEY_RESTART</td>
<td>0567</td>
<td>restart key</td>
</tr>
<tr>
<td>KEY_RESUME</td>
<td>0570</td>
<td>resume key</td>
</tr>
<tr>
<td>KEY_SAVE</td>
<td>0571</td>
<td>save key</td>
</tr>
<tr>
<td>KEY_SBEG</td>
<td>0572</td>
<td>shifted beginning key</td>
</tr>
<tr>
<td>KEY_SCANCEL</td>
<td>0573</td>
<td>shifted cancel key</td>
</tr>
<tr>
<td>KEY_SCOMMAND</td>
<td>0574</td>
<td>shifted command key</td>
</tr>
<tr>
<td>KEY_SCOPY</td>
<td>0575</td>
<td>shifted copy key</td>
</tr>
<tr>
<td>KEY_SCREATE</td>
<td>0576</td>
<td>shifted create key</td>
</tr>
<tr>
<td>KEY_SDC</td>
<td>0577</td>
<td>shifted delete char key</td>
</tr>
<tr>
<td>KEY(SDL)</td>
<td>0600</td>
<td>shifted delete line key</td>
</tr>
<tr>
<td>KEY_SELECT</td>
<td>0601</td>
<td>select key</td>
</tr>
<tr>
<td>KEY_SEND</td>
<td>0602</td>
<td>shifted end key</td>
</tr>
<tr>
<td>KEY_SEOL</td>
<td>0603</td>
<td>shifted clear line key</td>
</tr>
<tr>
<td>KEY_SEXIT</td>
<td>0604</td>
<td>shifted exit key</td>
</tr>
<tr>
<td>KEY_SFIN</td>
<td>0605</td>
<td>shifted find key</td>
</tr>
<tr>
<td>KEY_SHELP</td>
<td>0606</td>
<td>shifted help key</td>
</tr>
<tr>
<td>KEY_SHOME</td>
<td>0607</td>
<td>shifted home key</td>
</tr>
<tr>
<td>KEY_SIC</td>
<td>0610</td>
<td>shifted input key</td>
</tr>
<tr>
<td>KEY_SLEFT</td>
<td>0611</td>
<td>shifted left arrow key</td>
</tr>
<tr>
<td>KEY_SMESSAGE</td>
<td>0612</td>
<td>shifted message key</td>
</tr>
<tr>
<td>KEY_SMOVE</td>
<td>0613</td>
<td>shifted move key</td>
</tr>
<tr>
<td>KEY_SNEXT</td>
<td>0614</td>
<td>shifted next key</td>
</tr>
<tr>
<td>KEY_SOPTIONS</td>
<td>0615</td>
<td>shifted options key</td>
</tr>
<tr>
<td>KEY_SPREVIOUS</td>
<td>0616</td>
<td>shifted prev key</td>
</tr>
<tr>
<td>KEY_SPRINT</td>
<td>0617</td>
<td>shifted print key</td>
</tr>
<tr>
<td>KEY_SREDO</td>
<td>0620</td>
<td>shifted redo key</td>
</tr>
</tbody>
</table>

CURSES(3X) (Terminal Information Utilities)
The following variables may be used to add line-drawing characters to the screen with `waddch()`. When defined for the terminal, the variable will have the _A_ALTCHARSET_ bit turned on. Otherwise, the default character listed below will be stored in the variable. The names were chosen to be consistent with the DEC VT100 nomenclature.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Glyph Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS_ULCORNER</td>
<td>+</td>
<td>upper left corner</td>
</tr>
<tr>
<td>ACS_LLCORNER</td>
<td>+</td>
<td>lower left corner</td>
</tr>
<tr>
<td>ACS_URCORNER</td>
<td>+</td>
<td>upper right corner</td>
</tr>
<tr>
<td>ACS_LRCORNER</td>
<td>+</td>
<td>lower right corner</td>
</tr>
<tr>
<td>ACS_RTEE</td>
<td>+</td>
<td>right tee (-)</td>
</tr>
<tr>
<td>ACS_LTEE</td>
<td>+</td>
<td>left tee (←)</td>
</tr>
<tr>
<td>ACS_BTEE</td>
<td>+</td>
<td>bottom tee (_)</td>
</tr>
<tr>
<td>ACS_TTEE</td>
<td>+</td>
<td>top tee (↑)</td>
</tr>
<tr>
<td>ACS_HLINE</td>
<td>-</td>
<td>horizontal line</td>
</tr>
<tr>
<td>ACS_VLINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACS_PLUS</td>
<td>+</td>
<td>plus</td>
</tr>
<tr>
<td>ACS_S1</td>
<td>-</td>
<td>scan line 1</td>
</tr>
<tr>
<td>ACS_S9</td>
<td>-</td>
<td>scan line 9</td>
</tr>
<tr>
<td>ACS_DIAMOND</td>
<td>+</td>
<td>diamond</td>
</tr>
<tr>
<td>ACS_CKBOARD</td>
<td>:</td>
<td>checker board (stipple)</td>
</tr>
<tr>
<td>ACS_DEGREE</td>
<td>'</td>
<td>degree symbol</td>
</tr>
<tr>
<td>ACS_PMINUS</td>
<td>#</td>
<td>plus/minus</td>
</tr>
<tr>
<td>ACS_BULLET</td>
<td>o</td>
<td>bullet</td>
</tr>
<tr>
<td>ACS_LARROW</td>
<td>&lt;</td>
<td>arrow pointing left</td>
</tr>
<tr>
<td>ACS_RARROW</td>
<td>&gt;</td>
<td>arrow pointing right</td>
</tr>
<tr>
<td>ACS_DARROW</td>
<td>v</td>
<td>arrow pointing down</td>
</tr>
<tr>
<td>ACS_UARROW</td>
<td>^</td>
<td>arrow pointing up</td>
</tr>
<tr>
<td>ACS_BOARD</td>
<td>#</td>
<td>board of squares</td>
</tr>
<tr>
<td>ACS_LANTERN</td>
<td>#</td>
<td>lantern symbol</td>
</tr>
<tr>
<td>ACS_BLOCK</td>
<td>#</td>
<td>solid square block</td>
</tr>
</tbody>
</table>

**DIAGNOSTICS**

All routines return the integer **OK** upon successful completion and the integer **ERR** upon failure, unless otherwise noted in the preceding routine descriptions.
All macros return the value of their w version, except setscrreg(), wsetscrreg(), getsyx(), getyx(), getbegy(), getmaxyx(). For these macros, no useful value is returned.

Routines that return pointers always return (type *) NULL on error.

BUGS
Currently typeahead checking is done using a nodelay read followed by an ungetch() of any character that may have been read. Typeahead checking is done only if wgetch() has been called at least once. This will be changed when proper kernel support is available. Programs which use a mixture of their own input routines with curses input routines may wish to call typeahead(−1) to turn off typeahead checking.

The argument to napms() is currently rounded up to the nearest second.

draino (ms) only works for ms equal to 0.

WARNINGS
To use the new curses features, use the Release 3.0 version of curses on UNIX System Release 3.0. All programs that ran with System V Release 2 curses will run with System V Release 3.0. You may link applications with object files based on the Release 2 curses/terminfo with the Release 3.0 libcurses.a library. You may link applications with object files based on the Release 3.0 curses/terminfo with the Release 2 libcurses.a library, so long as the application does not use the new features in the Release 3.0 curses/terminfo.

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 plot(3X) code in a different source file than the curses(3X) code, and/or #undef move() and erase() in the plot(3X) code.

Between the time a call to initscr() and endwin() has been issued, use only the routines in the curses library to generate output. Using system calls or the "standard I/O package" (see stdio(3S)) for output during that time can cause unpredictable results.

SEE ALSO
cc(1), ld(1), ioctl(2), plot(3X), putc(3S), scanf(3S), stdio(3S), system(3S), vprintf(3S), profile(4), term(4), terminfo(4), varargs(5), termio(7), tty(7) in the System Administrator's Reference Manual. Chapter 10 of the Programmer's Guide.
NAME

directory: opendir, readdir, telldir, seekdir, rewinddir, closedir — directory operations

SYNOPSIS

#include <sys/types.h>
#include <dirent.h>
RTL *opendir (filename)
char *filename;
struct dirent *readdir (dirp)
DIR *dirp;
long telldir (dirp)
DIR *dirp;
void seekdir (dirp, loc)
DIR *dirp;
long loc;
void rewinddir (dirp)
DIR *dirp;
void closedir(dirp)
DIR *dirp;

DESCRIPTION

Opendir opens the directory named by filename and associates a directory stream with it. Opendir returns a pointer to be used to identify the directory stream in subsequent operations. The pointer NULL is returned if filename cannot be accessed or is not a directory, or if it cannot malloc(3X) enough memory to hold a DIR structure or a buffer for the directory entries.

Readdir returns a pointer to the next active directory entry. No inactive entries are returned. It returns NULL upon reaching the end of the directory or upon detecting an invalid location in the directory.

Telldir returns the current location associated with the named directory stream.

Seekdir sets the position of the next readdir operation on the directory stream. The new position reverts to the one associated with the directory stream when the telldir operation from which loc was obtained was performed. Values returned by telldir are good only if the directory has not changed due to compaction or expansion. This is not a problem with System V, but it may be with some file system types.

Rewinddir resets the position of the named directory stream to the beginning of the directory.

Closedir closes the named directory stream and frees the DIR structure.

The following errors can occur as a result of these operations.
opendir:

[ENOTDIR] A component of filename is not a directory.

[EACCES] A component of filename denies search permission.

[EMFILE] The maximum number of file descriptors are currently open.

[EFAULT] Filename points outside the allocated address space.

readdir:

[ENOENT] The current file pointer for the directory is not located at a valid entry.

[EBADF] The file descriptor determined by the DIR stream is no longer valid. This results if the DIR stream has been closed.

telldir, seekdir, and closedir:

[EBADF] The file descriptor determined by the DIR stream is no longer valid. This results if the DIR stream has been closed.

EXAMPLE

Sample code which searches a directory for entry name:

```c
    dirp = opendir( "." );
    while ( (dp = readdir(dirp)) != NULL )
        if ( strcmp(dp->d_name, name) == 0 ) {
            closedir(dirp);
            return FOUND;
        }
    closedir(dirp);
    return NOT_FOUND;
```

SEE ALSO

getdents(2), dirent(4).

WARNINGS

Rewinddir is implemented as a macro, so its function address cannot be taken.

NAME

ldahread — read the archive header of a member of an archive file

SYNOPSIS

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

int ldahread (ldptr, arhead)
    LDFILE *ldptr;
    ARCHDR *arhead;
```
DESCRIPTION
If TYPE(ldptr) is the archive file magic number, ldahread reads the archive header of the common object file currently associated with ldptr into the area of memory beginning at arhead.

ldahread returns SUCCESS or FAILURE. ldahread will fail if TYPE(ldptr) does not represent an archive file, or if it cannot read the archive header.

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

SEE ALSO
ldclose(3X), ldopen(3X), ldfcn(4), ar(4).
NAME
ldclose, ldaclose — close a common object file

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

int ldclose (ldptr)
LDFILE *ldptr;

int ldaclose (ldptr)
LDFILE *ldptr;

DESCRIPTION
Ldopen(3X) and ldclose are designed to provide uniform access to both simple
object files and object files that are members of archive files. Thus an archive of
common object files can be processed as if it were a series of simple common
object files.

If TYPE(ldptr) does not represent an archive file, ldclose will close the file and
free the memory allocated to the LDFILE structure associated with ldptr. If
TYPE(ldptr) is the magic number of an archive file, and if there are any more
files in the archive, ldclose will reinitialize OFFSET(ldptr) to the file address of the
next archive member and return FAILURE. The LDFILE structure is prepared for
a subsequent ldopen(3X). In all other cases, ldclose returns SUCCESS.

Ldaclose closes the file and frees the memory allocated to the LDFILE structure
associated with ldptr regardless of the value of TYPE(ldptr). Ldaclose always
returns SUCCESS. The function is often used in conjunction with ldaopen.

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

SEE ALSO
fclose(3S), ldopen(3X), ldfcn(4).
NAME

ldfhread — read the file header of a common object file

SYNOPSIS

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

int ldfhread (ldptr, filehead)
LDFILE *ldptr;
FILHDR *filehead;

DESCRIPTION

ldfhread reads the file header of the common object file currently associated with
ldptr into the area of memory beginning at filehead.

ldfhread returns SUCCESS or FAILURE. ldfhread will fail if it cannot read the file
header.

In most cases the use of ldfhread can be avoided by using the macro
HEADER(ldptr) defined in ldfcn.h [see ldfcn (4)]. The information in any field,
fieldname, of the file header may be accessed using HEADER(ldptr).fieldname.

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

SEE ALSO

ldclose(3X), ldopen(3X), ldfcn(4).
NAME

**ldgetname** – retrieve symbol name for common object file symbol table entry

SYNOPSIS

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

char *ldgetname (ldptr, symbol)
LDFILE *ldptr;
SYMENT *symbol;
```

DESCRIPTION

**ldgetname** returns a pointer to the name associated with **symbol** as a string. The string is contained in a static buffer local to **ldgetname** that is overwritten by each call to **ldgetname**, and therefore must be copied by the caller if the name is to be saved.

**ldgetname** can be used to retrieve names from object files without any backward compatibility problems. **ldgetname** will return NULL (defined in **stdio.h**) for an object file if the name cannot be retrieved. This situation can occur:

- if the “string table” cannot be found,
- if not enough memory can be allocated for the string table,
- if the string table appears not to be a string table (for example, if an auxiliary entry is handed to **ldgetname** that looks like a reference to a name in a nonexistent string table), or
- if the name’s offset into the string table is past the end of the string table.

Typically, **ldgetname** will be called immediately after a successful call to **ldtbread** to retrieve the name associated with the symbol table entry filled by **ldtbread**.

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

SEE ALSO

**ldclose(3X)**, **ldopen(3X)**, **ldtbread(3X)**, **ldtbseek(3X)**, **ldfcn(4)**.
NAME
ldlread, ldlinit, ldlitem — manipulate line number entries of a common object file
function

SYNOPSIS
#include <stdio.h>
#include <filehdr.h>
#include <linenum.h>
#include <ldfcn.h>

int ldlread(ldptr, fcnindx, linenum, linent)
LDFILE *ldptr;
long fcnindx;
unsigned short linenum;
LINENO *linent;

int ldlinit(ldptr, fcnindx)
LDFILE *ldptr;
long fcnindx;

int ldlitem(ldptr, linenum, linent)
LDFILE *ldptr;
unsigned short linenum;
LINENO *linent;

DESCRIPTION
ldlread searches the line number entries of the common object file currently
associated with ldptr. ldlread begins its search with the line number entry for
the beginning of a function and confines its search to the line numbers associ­
ated with a single function. The function is identified by fcnindx, the index of its
entry in the object file symbol table. ldlread reads the entry with the smallest
line number equal to or greater than linenum into the memory beginning at
linent.

Ldlinit and ldlitem together perform exactly the same function as ldlread. After
an initial call to ldlread or ldlinit, ldlitem may be used to retrieve a series of line
number entries associated with a single function. Ldlinit simply locates the line
number entries for the function identified by fcnindx. Ldlitem finds and reads the
entry with the smallest line number equal to or greater than linenum into the
memory beginning at linent.

ldlread, ldlinit, and ldlitem each return either SUCCESS or FAILURE. ldlread will
fail if there are no line number entries in the object file, if fcnindx does not index
a function entry in the symbol table, or if it finds no line number equal to or
greater than linenum. Ldlinit will fail if there are no line number entries in the
object file or if fcnindx does not index a function entry in the symbol table.
Ldlitem will fail if it finds no line number equal to or greater than linenum.

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

SEE ALSO
ldclose(3X), ldopen(3X), ldtbindex(3X), ldfcn(4).
NAME

ldlseek, ldnlseek — seek to line number entries of a section of a common object file

SYNOPSIS

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

int ldlseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnlseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

DESCRIPTION

`ldlseek` seeks to the line number entries of the section specified by `sectindx` of the common object file currently associated with `ldptr`.

`ldnlseek` seeks to the line number entries of the section specified by `sectname`.

`ldlseek` and `ldnlseek` return SUCCESS or FAILURE. `ldlseek` will fail if `sectindx` is greater than the number of sections in the object file; `ldnlseek` will fail if there is no section name corresponding with `sectname`. Either function will fail if the specified section has no line number entries or if it cannot seek to the specified line number entries.

Note that the first section has an index of one.

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

SEE ALSO

ldclose(3X), ldopen(3X), ldshread(3X), ldfcn(4).
NAME
Idohseek — seek to the optional file header of a common object file

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

int Idohseek (ldptr)
LDFILE *ldptr;

DESCRIPTION
Idohseek seeks to the optional file header of the common object file currently
associated with ldptr.

Idohseek returns SUCCESS or FAILURE. Idohseek will fail if the object file has no
optional header or if it cannot seek to the optional header.

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

SEE ALSO
IdcloselX), Idopen(X), ldthread(X), ld(4).
NAME
ldopen, ldaopen — open a common object file for reading

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

LDFILE *ldopen (filename, ldptr)
char *filename;
LDFILE *ldptr;

LDFILE *ldaopen (filename, oldptr)
char *filename;
LDFILE *oldptr;

DESCRIPTION
ldopen and ldclose(3X) are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of common object files can be processed as if it were a series of simple common object files.

If ldptr has the value NULL, then ldopen will open filename and allocate and initialize the LDFILE structure, and return a pointer to the structure to the calling program.

If ldptr is valid and if TYPE(ldptr) is the archive magic number, ldopen will reinitialize the LDFILE structure for the next archive member of filename.

ldopen and ldclose(3X) are designed to work in concert. Ldclose will return FAILURE only when TYPE(ldptr) is the archive magic number and there is another file in the archive to be processed. Only then should ldopen be called with the current value of ldptr. In all other cases, in particular whenever a new filename is opened, ldopen should be called with a NULL ldptr argument.

The following is a prototype for the use of ldopen and ldclose(3X).

/* for each filename to be processed */
ldptr = NULL;
do {
   if ( (ldptr = ldopen(filename, ldptr)) != NULL )
   {
      /* check magic number */
      /* process the file */
   }
} while (ldclose(ldptr) == FAILURE);

If the value of oldptr is not NULL, ldaopen will open filename anew and allocate and initialize a new LDFILE structure, copying the TYPE, OFFSET, and HEADER fields from oldptr. Ldaopen returns a pointer to the new LDFILE structure. This new pointer is independent of the old pointer, oldptr. The two pointers may be used concurrently to read separate parts of the object file. For example, one pointer may be used to step sequentially through the relocation information, while the other is used to read indexed symbol table entries.
Both `ldopen` and `ldaopen` open `filename` for reading. Both functions return `NULL` if `filename` cannot be opened, or if memory for the `LDFILE` structure cannot be allocated. A successful open does not insure that the given file is a common object file or an archived object file.

The program must be loaded with the object file access routine library `libId.a`.

SEE ALSO
`fopen(3S), ldclose(3X), ldfcn(4)`. 
NAME

ldrseek, ldnrseek — seek to relocation entries of a section of a common object file

SYNOPSIS

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

int ldrseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnrseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

DESCRIPTION

`ldrseek` seeks to the relocation entries of the section specified by `sectindx` of the common object file currently associated with `ldptr`.

`ldnrseek` seeks to the relocation entries of the section specified by `sectname`.

`ldrseek` and `ldnrseek` return SUCCESS or FAILURE. `ldrseek` will fail if `sectindx` is greater than the number of sections in the object file; `ldnrseek` will fail if there is no section name corresponding with `sectname`. Either function will fail if the specified section has no relocation entries or if it cannot seek to the specified relocation entries.

Note that the first section has an index of one.

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

SEE ALSO

ldclose(3X), ldopen(3X), ldshread(3X), ldfcn(4).
NAME
ldshread, ldnshread — read an indexed/named section header of a common object file

SYNOPSIS

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

int ldshread (ldptr, sectindx, secthead)
LDFILE *ldptr;
unsigned short sectindx;
SCNHDR *secthead;

int ldnshread (ldptr, sectname, secthead)
LDFILE *ldptr;
char *sectname;
SCNHDR *secthead;

DESCRIPTION

ldshread reads the section header specified by sectindx of the common object file currently associated with ldptr into the area of memory beginning at secthead.

ldnshread reads the section header specified by sectname into the area of memory beginning at secthead.

ldshread and ldnshread return SUCCESS or FAILURE. ldshread will fail if sectindx is greater than the number of sections in the object file; ldnshread will fail if there is no section name corresponding with sectname. Either function will fail if it cannot read the specified section header.

Note that the first section header has an index of one.

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

SEE ALSO
ldclose(3X), ldopen(3X), ldfcn(4).
NAME
ldsseek, ldnsseek — seek to an indexed/named section of a common object file

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

int ldsseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnsseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;

DESCRIPTION
ldsseek seeks to the section specified by sectindx of the common object file currently associated with ldptr.
Ldnsseek seeks to the section specified by sectname.
ldsseek and ldnsseek return SUCCESS or FAILURE. ldsseek will fail if sectindx is greater than the number of sections in the object file; ldnsseek will fail if there is no section name corresponding with sectname. Either function will fail if there is no section data for the specified section or if it cannot seek to the specified section.

Note that the first section has an index of one.
The program must be loaded with the object file access routine library libld.a.

SEE ALSO
Idclose(3X), Idopen(3X), ldshread(3X), ldfcn(4).
NAME
ldtbindex — compute the index of a symbol table entry of a common object file

SYNOPSIS
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

long ldtbindex (ldptr)
LDFILE *ldptr;

DESCRIPTION
ldtbindex returns the (long) index of the symbol table entry at the current position of the common object file associated with ldptr.

The index returned by ldtbindex may be used in subsequent calls to ldtbread(3X). However, since ldtbindex returns the index of the symbol table entry that begins at the current position of the object file, if ldtbindex is called immediately after a particular symbol table entry has been read, it will return the index of the next entry.

ldtbindex will fail if there are no symbols in the object file, or if the object file is not positioned at the beginning of a symbol table entry.

Note that the first symbol in the symbol table has an index of zero.

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

SEE ALSO
ldclose(3X), ldopen(3X), ldtbread(3X), ldtbseek(3X), ldfcn(4).
NAME
ldtbread — read an indexed symbol table entry of a common object file

SYNOPSIS

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

int ldtbred (ldptr, symindex, symbol)
LDFILE *ldptr;
long symindex;
SYMENT *symbol;
```

DESCRIPTION

ldtbread reads the symbol table entry specified by symindex of the common object file currently associated with ldptr into the area of memory beginning at symbol.

ldtbread returns SUCCESS or FAILURE. ldtbred will fail if symindex is greater than or equal to the number of symbols in the object file, or if it cannot read the specified symbol table entry.

Note that the first symbol in the symbol table has an index of zero.

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

SEE ALSO

ldclose(3X), ldopen(3X), ldtbseek(3X), ldgetname(3X), ldfcn(4).
NAME
ldtbseek — seek to the symbol table of a common object file

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

int ldtbseek (ldptr)
LDFILE *ldptr;

DESCRIPTION
ldtbseek seeks to the symbol table of the common object file currently associated
with ldptr.

ldtbseek returns SUCCESS or FAILURE. ldtbseek will fail if the symbol table has
been stripped from the object file, or if it cannot seek to the symbol table.
The program must be loaded with the object file access routine library libld.a.

SEE ALSO
ldclose(3X), ldopen(3X), ldtbread(3X), ldfcn(4).
NAME
libwindows - windowing terminal function library

SYNOPSIS
cc [flag ...] file ... -lwindows [library ...]
int cntlfd, fd
int chan
int origin_x, origin_y, corner_x, corner_y
char *command

cntlfd = openagent ( )
chan = New (cntlfd, origin_x, origin_y, corner_x, corner_y)
chan = Newlayer (cntlfd, origin_x, origin_y, corner_x, corner_y)
fd = openchan (chan)
Runlayer (chan, command)
Current (cntlfd, chan)
Delete (cntlfd, chan)
Top (cntlfd, chan)
Bottom (cntlfd, chan)
Move (cntlfd, chan, origin_x, origin_y)
Reshape (cntlfd, chan, origin_x, origin_y, corner_x, corner_y)
Exit (cntlfd)

DESCRIPTION
This library of routines enables a program running on a host UNIX system to
perform windowing terminal functions (see layers(1)).

The openagent() routine opens the control channel of the xt(7) channel group to
which the calling process belongs. Upon successful completion, openagent()
returns a file descriptor, cntlfd, that can be passed to any of the other libwindows
routines except openchan() and Runlayer(). (cntlfd can also be passed to
close(2).) Otherwise, the value -1 is returned.

The New() routine creates a new layer with a separate shell. The origin_x,
origin_y, corner_x, and corner_y arguments are the coordinates of the layer re­
tangle. If all the coordinate arguments are 0, the user must define the layer’s
rectangle interactively. The layer appears on top of any overlapping layers. The
layer is not made current (i.e., the keyboard is not attached to the new layer).
Upon successful completion, New() returns the xt(7) channel number associated
with the layer. Otherwise, the value -1 is returned.

The Newlayer() routine creates a new layer without executing a separate shell.
Otherwise it is identical to New(), described above.

The openchan() routine opens the channel argument chan which is obtained
from the New() or Newlayer() routine. Upon successful completion, open­
chan() returns a file descriptor that can be used as input to write(2) or close(2).
Otherwise, the value -1 is returned.

The **Runlayer()** routine runs the specified *command* in the layer associated with the channel argument *chan*. Any processes currently attached to this layer will be killed, and the new process will have the environment of the *layers(1)* process.

The **Current()** routine makes the layer associated with the channel argument *chan* current (i.e., attached to the keyboard).

The **Delete()** routine deletes the layer associated with the channel argument *chan* and kills all host processes associated with the layer.

The **Top()** routine makes the layer associated with the channel argument *chan* appear on top of all overlapping layers.

The **Bottom()** routine puts the layer associated with the channel argument *chan* under all overlapping layers.

The **Move()** routine moves the layer associated with the channel argument *chan* from its current screen location to a new screen location at the origin point (*origin_x*, *origin_y*). The size and contents of the layer are maintained.

The **Reshape()** routine reshapes the layer associated with the channel argument *chan*. The arguments *origin_x*, *origin_y*, *corner_x*, and *corner_y* are the new coordinates of the layer rectangle. If all the coordinate arguments are 0, the user is allowed to define the layer's rectangle interactively.

The **Exit()** routine causes the *layers(1)* program to exit, killing all processes associated with it.

**RETURN VALUE**

Upon successful completion, **Runlayer()**, **Current()**, **Delete()**, **Top()**, **Bottom()**, **Move()**, **Reshape()**, and **Exit()** return a 0, while **openagent()**, **New()**, **Newlayer()**, and **openchan()** return values as described above under each routine. If an error occurs, -1 is returned.

**FILES**

/usr/lib/libwindows.a  windowing terminal function library

**NOTE**

The values of layer rectangle coordinates are dependent on the type of terminal. This dependency affects the routines that pass layer rectangle coordinates: **Move()**, **New()**, **Newlayer()**, and **Reshape()**. Some terminals will expect these numbers to be passed as character positions (bytes); others will expect the information to be in pixels (bits).

For example, for the AT&T Teletype 5620 DMD terminal, **New()**, **Newlayer()**, and **Reshape()** take minimum values of 8 (pixels) for *origin_x* and *origin_y* and maximum values of 792 (pixels) for *corner_x* and 1016 (pixels) for *corner_y*. In addition, the minimum layer size is 28 by 28 pixels and the maximum layer size is 784 by 1008 pixels.

**SEE ALSO**

`close(2)`, `jagent(5)`, `write(2)`.

*layers(1)* in the *User's Reference Manual*.

*xt(7)* in the *System Administrator's Reference Manual*. 
NAME
logname — return login name of user

SYNOPSIS
char *logname();

DESCRIPTION
logname returns a pointer to the null-terminated login name; it extracts the
LOGNAME environment variable from the user's environment.

This routine is kept in /lib/libPW.a.

FILES
/etc/profile

SEE ALSO
getenv(3C), profile(4), environ(5).

CAVEATS
The return values point to static data whose content is overwritten by each call.
This method of determining a login name is subject to forgery.
NAME
malloc, free, realloc, calloc, mallopt, mallinfo — fast main memory allocator

SYNOPSIS
#include <malloc.h>
char *malloc (size)
unsigned size;
void free (ptr)
char *ptr;
char *realloc (ptr, size)
char *ptr;
unsigned size;
char *calloc (nelem, elsize)
unsigned nelem, elsize;
int mallopt (cmd, value)
int cmd, value;
struct mallinfo mallinfo();

DESCRIPTION
malloc and free provide a simple general-purpose memory allocation package, which runs considerably faster than the malloc(3C) package. It is found in the library "malloc", and is loaded if the option "-lmalloc" is used with cc(1) or ld(1).

malloc returns a pointer to a block of at least size bytes suitably aligned for any use.

The argument to free is a pointer to a block previously allocated by malloc; after free is performed this space is made available for further allocation, and its contents have been destroyed (but see mallopt below for a way to change this behavior).

Undefined results will occur if the space assigned by malloc is overrun or if some random number is handed to free.

Realloc changes the size of the block pointed to by ptr to size bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

Calloc allocates space for an array of nelem elements of size elsize. The space is initialized to zeros.

Mallopt provides for control over the allocation algorithm. The available values for cmd are:

M_MXFAST Set maxfast to value. The algorithm allocates all blocks below the size of maxfast in large groups and then doles them out very quickly. The default value for maxfast is 24.

M_NLBLKS Set numlblks to value. The above mentioned “large groups” each contain numlblks blocks. Numlblks must be greater than 0. The default value for numlblks is 100.
M_GRAIN

Set grain to value. The sizes of all blocks smaller than maxfast are considered to be rounded up to the nearest multiple of grain. Grain must be greater than 0. The default value of grain is the smallest number of bytes which will allow alignment of any data type. Value will be rounded up to a multiple of the default when grain is set.

M_KEEP

Preserve data in a freed block until the next malloc, realloc, or calloc. This option is provided only for compatibility with the old version of malloc and is not recommended.

These values are defined in the <malloc.h> header file.

Malloc may be called repeatedly, but may not be called after the first small block is allocated.

Mallinfo provides instrumentation describing space usage. It returns the structure:

```
struct mallinfo {
    int arena;  /* total space in arena */
    int ordblks; /* number of ordinary blocks */
    int smblks; /* number of small blocks */
    int hblkhd; /* space in holding block headers */
    int hblks; /* number of holding blocks */
    int usmblks; /* space in small blocks in use */
    int fsmblks; /* space in free small blocks */
    int uordblks; /* space in ordinary blocks in use */
    int fordblks; /* space in free ordinary blocks */
    int keepcost; /* space penalty if keep option */
                   /* is used */
}
```

This structure is defined in the <malloc.h> header file.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

SEE ALSO

brk(2), malloc(3C).

DIAGNOSTICS

malloc, realloc and calloc return a NULL pointer if there is not enough available memory. When realloc returns NULL, the block pointed to by ptr is left intact. If malloc is called after any allocation or if cmd or value are invalid, non-zero is returned. Otherwise, it returns zero.

WARNINGS

This package usually uses more data space than malloc(3C).
The code size is also bigger than malloc(3C).
Note that unlike malloc(3C), this package does not preserve the contents of a block when it is freed, unless the M_KEEP option of malloc is used.
Undocumented features of malloc(3C) have not been duplicated.
NAME
plot – graphics interface subroutines

SYNOPSIS
openpl ()
erase ()
label (s)
char *s;
line (x1, y1, x2, y2)
int x1, y1, x2, y2;
circle (x, y, r)
int x, y, r;
arc (x, y, x0, y0, x1, y1)
int x, y, x0, y0, x1, y1;
move (x, y)
int x, y;
cont (x, y)
int x, y;
point (x, y)
int x, y;
linemod (s)
char *s;
space (x0, y0, x1, y1)
int x0, y0, x1, y1;
closepl ()

DESCRIPTION
These subroutines generate graphic output in a relatively device-independent manner. Space must be used before any of these functions to declare the amount of space necessary [see plot(4)]. Openpl must be used before any of the others to open the device for writing. Closepl flushes the output.

Circle draws a circle of radius r with center at the point (x, y).

Arc draws an arc of a circle with center at the point (x, y) between the points (x0, y0) and (x1, y1).

String arguments to label and linemod are terminated by nulls and do not contain new-lines.

See plot(4) for a description of the effect of the remaining functions.

The library files listed below provide several flavors of these routines.

FILES
LIBDIR/libplot.a produces output for tplot(1G) filters
LIBDIR/lib300.pa for DASI 300
LIBDIR/lib300.a for DASI 300s
LIBDIR/lib450.a for DASI 450
LIBDIR/lib4014.a for TEKTRONIX 4014
LIBDIR usually /usr/lib

SEE ALSO
plot(4),

WARNINGS
In order to compile a program containing these functions in file.c it is necessary
to use "cc file.c -lplot".
In order to execute it, it is necessary to use "a.out tplot".
The above routines use <stdio.h>, which causes them to increase the size of
programs, not otherwise using standard I/O more than might be expected.
NAME
regcmp, regex — compile and execute regular expression

SYNOPSIS

\[
\begin{align*}
\text{char } *\text{regcmp} & \text{ (string1 [, string2, ...], (char *)0)} \\
\text{char } *\text{string1, *string2, ... ;} \\
\text{char } *\text{regex (re, subject[, retO, ...])} \\
\text{char } *\text{re, *subject, *retO, ... ;} \\
\text{extern char } *\_\text{locI};
\end{align*}
\]

DESCRIPTION

regcmp compiles a regular expression (consisting of the concatenated arguments) and returns a pointer to the compiled form. Malloc(3C) is used to create space for the compiled form. It is the user’s responsibility to free unneeded space so allocated. A NULL return from regcmp indicates an incorrect argument. regcmp(1) has been written to generally preclude the need for this routine at execution time.

Regex executes a compiled pattern against the subject string. Additional arguments are passed to receive values back. Regex returns NULL on failure or a pointer to the next unmatched character on success. A global character pointer _locI points to where the match began. regcmp and regex were mostly borrowed from the editor, ed(1); however, the syntax and semantics have been changed slightly. The following are the valid symbols and their associated meanings.

\[
\begin{align*}
[ ]^*.^* & \quad \text{These symbols retain their meaning in ed(1).} \\
$ & \quad \text{Matches the end of the string; \text{n matches a new-line.} } \\
- & \quad \text{Within brackets the minus means through. For example, [a–z] is equivalent to [abcd...xyz]. The – can appear as itself only if used as the first or last character. For example, the character class expression } \\
& \quad \text{[\[\]–\]} \text{ matches the characters } \text{] and –.} \\
+ & \quad \text{A regular expression followed by + means one or more times. For example, [0–9]+ is equivalent to [0–9] [0–9]*.} \\
\{m\} & \quad \text{Integer values enclosed in } \{\} \text{ indicate the number of times the preceding} \\
\{m,u\} & \quad \text{regular expression is to be applied. The value } m \text{ is the minimum number and } u \text{ is a number, less than 256, which is the maximum. If only } m \text{ is present (e.g., } \{m\} \text{), it indicates the exact number of times the} \\
& \quad \text{regular expression is to be applied. The value } \{m,u\} \text{ is analogous to } \\
& \quad \text{\{m,infinity\}. The plus (+) and star (*) operations are equivalent to } \{1\} \\
& \quad \text{and } \{0\} \text{ respectively.} \\
(\ldots)$n & \quad \text{The value of the enclosed regular expression is to be returned. The} \\
& \quad \text{value will be stored in the } (n+1)\text{th argument following the subject} \\
& \quad \text{argument. At most ten enclosed regular expressions are allowed. Regex} \\
& \quad \text{makes its assignments unconditionally.}
\end{align*}
\]
Parentheses are used for grouping. An operator, e.g., *, +, {, can work on a single character or a regular expression enclosed in parentheses. For example, (a*(cb+))*$0.

By necessity, all the above defined symbols are special. They must, therefore, be escaped with a \ (backslash) to be used as themselves.

**EXAMPLES**

Example 1:
```c
char *cursor, *newcursor, *ptr;
...
newcursor = regex((ptr = regcmp("\n", (char *)0)), cursor);
free(ptr);
```
This example will match a leading new-line in the subject string pointed at by cursor.

Example 2:
```c
char ret0[9];
char *newcursor, *name;
...
name = regcmp("([A-Za-z][A-za-z0-9][0,7])0", (char *)0);
newcursor = regex(name, "012Testing345", ret0);
```
This example will match through the string "Testing3" and will return the address of the character after the last matched character (the '4'). The string "Testing3" will be copied to the character array ret0.

Example 3:
```c
#include "file.i"
char *string, *newcursor;
...
newcursor = regex(name, string);
```
This example applies a precompiled regular expression in file.i [see regcmp(1)] against string.

These routines are kept in /lib/libPW.a.

**SEE ALSO**
regcmp(1), malloc(3C).

**BUGS**
The user program may run out of memory if regcmp is called iteratively without freeing the vectors no longer required.
NAME
  abort – terminate Fortran program

SYNOPSIS
  call abort ( )

DESCRIPTION
  abort terminates the program which calls it, closing all open files truncated to the
current position of the file pointer. The abort usually results in a core dump.

DIAGNOSTICS
  When invoked, abort prints “Fortran abort routine called” on the standard error
output. The shell prints the message “abort - core dumped” if a core dump
results.

SEE ALSO
  abort(3C).
NAME
abs, iabs, dabs, cabs, zabs — Fortran absolute value

SYNOPSIS

integer i1, i2
real r1, r2
double precision dp1, dp2
complex cx1, cx2
double complex dx1, dx2
r2 = abs(r1)
i2 = iabs(i1)
i2 = abs(i1)
dp2 = dabs(dp1)
dp2 = abs(dp1)
cx2 = cabs(cx1)
cx2 = abs(cx1)
dx2 = zabs(dx1)
dx2 = abs(dx1)

DESCRIPTION
abs is the family of absolute value functions. iabs returns the integer absolute value of its integer argument. dabs returns the double-precision absolute value of its double-precision argument. cabs returns the complex absolute value of its complex argument. zabs returns the double-complex absolute value of its double-complex argument. The generic form abs returns the type of its argument.

SEE ALSO
floor(3M).
NAME
acos, dacos — Fortran arccosine intrinsic function

SYNOPSIS
real r1, r2
double precision dp1, dp2
r2 = acos(r1)
dp2 = dacos(dp1)
dp2 = acos(dp1)

DESCRIPTION
acos returns the real arccosine of its real argument. Dacos returns the double-precision arccosine of its double-precision argument. The generic form acos may be used with impunity as its argument will determine the type of the returned value.

SEE ALSO
trig(3M).
NAME
aimag, dimag — Fortran imaginary part of complex argument

SYNOPSIS
real r
complex cxr
double precision dp
double complex cxd
r = aimag(cxr)

DESCRIPTION
aimag returns the imaginary part of its single-precision complex argument.
Dimag returns the double-precision imaginary part of its double-complex argument.
NAME
aint, dint — Fortran integer part intrinsic function

SYNOPSIS
real r1, r2
double precision dp1, dp2
r2 = aint(r1)
dp2 = dint(dp1)
dp2 = aint(dp1)

DESCRIPTION
aint returns the truncated value of its real argument in a real. Dint returns the truncated value of its double-precision argument as a double-precision value. aint may be used as a generic function name, returning either a real or double-precision value depending on the type of its argument.
NAME
asin, dasin — Fortran arcsine intrinsic function

SYNOPSIS
real r1, r2
double precision dp1, dp2
r2 = asin(r1)
dp2 = dasin(dp1)
dp2 = asin(dp1)

DESCRIPTION
asin returns the real arcsine of its real argument. Dasin returns the double-
precision arcsine of its double-precision argument. The generic form asin may
be used with impunity as it derives its type from that of its argument.

SEE ALSO
trig(3M).
NAME
atan, datan – Fortran arctangent intrinsic function

SYNOPSIS

    real r1, r2
    double precision dp1, dp2
    r2 = atan(r1)
    dp2 = datan(dp1)
    dp2 = atanh(dp1)

DESCRIPTION

atan returns the real arctangent of its real argument. Datan returns the double-
precision arctangent of its double-precision argument. The generic form atan
may be used with a double-precision argument returning a double-precision
value.

SEE ALSO

trig(3M).
NAME
atan2, datan2 — Fortran arctangent intrinsic function

SYNOPSIS
real r1, r2, r3
double precision dp1, dp2, dp3
r3 = atan2(r1, r2)
dp3 = datan2(dp1, dp2)
dp3 = atan2(dp1, dp2)

DESCRIPTION
atan2 returns the arctangent of \( \arg_1/\arg_2 \) as a real value. Datan2 returns the double-precision arctangent of its double-precision arguments. The generic form atan2 may be used with impunity with double-precision arguments.

SEE ALSO
trig(3M).
NAME

bool: and, or, xor, not, lshift, rshift – Fortran Bitwise Boolean functions

SYNOPSIS

integer i, j, k
real a, b, c
k = and(i, j)
c = or(a, b)
j = xor(i, a)
j = not(i)
k = lshift(i, j)
k = rshift(i, j)

DESCRIPTION

The generic intrinsic Boolean functions and, or and xor return the value of the binary operations on their arguments. Not is a unary operator returning the one’s complement of its argument. Lshift and rshift return the value of the first argument shifted left or right, respectively, the number of times specified by the second (integer) argument.

While it is recommended that Boolean functions be used only on integer data, these functions are generic; that is, they are defined for all data types as arguments and return values. Where required, the compiler generates appropriate type conversions. However, when the functions are not used with integer data, the results are unpredictable.

BUGS

The implementation of the shift functions may cause large shift values to deliver weird results.

SEE ALSO

mil(3F).
NAME
conjg, dconjg — Fortran complex conjugate intrinsic function

SYNOPSIS
complex cx1, cx2
double complex dx1, dx2
cx2 = conjg(cx1)
dx2 = dconjg(dx1)

DESCRIPTION
conjg returns the complex conjugate of its complex argument. Dconjg returns the double-complex conjugate of its double-complex argument.
NAME

\texttt{cos, dcos, ccos} – Fortran cosine intrinsic function

SYNOPSIS

\begin{verbatim}
real r1, r2
double precision dp1, dp2
complex cx1, cx2
\end{verbatim}

\begin{verbatim}
r2 = \texttt{cos}(r1)
dp2 = \texttt{dcos}(dp1)
dp2 = \texttt{cos}(dp1)
cx2 = \texttt{ccos}(cx1)
cx2 = \texttt{cos}(cx1)
\end{verbatim}

DESCRIPTION

\texttt{cos} returns the real cosine of its real argument. \texttt{Dcos} returns the double-precision cosine of its double-precision argument. \texttt{Ccos} returns the complex cosine of its complex argument. The generic form \texttt{cos} may be used with impunity as its returned type is determined by that of its argument.

SEE ALSO

\texttt{trig(3M)}. 

\texttt{485}
COSH(3F)  (FORTRAN Programming Language Utilities)  COSH(3F)

NAME
  cosh, dcosh — Fortran hyperbolic cosine intrinsic function

SYNOPSIS
  real r1, r2
  double precision dp1, dp2
  r2 = cosh(r1)
  dp2 = dcosh(dp1)
  dp2 = cosh(dp1)

DESCRIPTION
  *cosh* returns the real hyperbolic cosine of its real argument. *Dcosh* returns the
double-precision hyperbolic cosine of its double-precision argument. The gen-
eric form *cosh* may be used to return the hyperbolic cosine in the type of its
argument.

SEE ALSO
  sinh(3M).
NAME
dim, ddim, idim — positive difference intrinsic functions

SYNOPSIS

integer a1, a2, a3
a3 = idim(a1, a2)

real a1, a2, a3
a3 = dim(a1, a2)

double precision a1, a2, a3
a3 = ddim(a1, a2)

DESCRIPTION
These functions return:
a1—a2  if a1 > a2
0      if a1 <= a2
NAME
dprod – double precision product intrinsic function

SYNOPSIS
real a1, a2
double precision a3
a3 = dprod(a1, a2)

DESCRIPTION
Dprod returns the double precision product of its real arguments.
NAME
exp, dexp, cexp – Fortran exponential intrinsic function

SYNOPSIS

real r1, r2
double precision dp1, dp2
complex cx1, cx2

r2 = exp(r1)
dp2 = dexp(dp1)
dp2 = exp(dp1)
cx2 = cexp(cx1)
cx2 = exp(cx1)

DESCRIPTION
exp returns the real exponential function $e^x$ of its real argument. Dexp returns the double-precision exponential function of its double-precision argument. Cexp returns the complex exponential function of its complex argument. The generic function exp becomes a call to dexp or cexp as required, depending on the type of its argument.

SEE ALSO
exp(3M).
NAME
ftype: int, ifix, idint, real, float, sngl, dble, cmplx, dcmplx, ichar, char — explicit Fortran type conversion

SYNOPSIS
integer i, j
real r, s
double precision dp, dq
complex cx
double complex dcx
character*1 ch
i = int(r)
i = int(dp)
i = int(cx)
i = int(dcx)
i = ifix(r)
i = idint(dp)
r = real(i)
r = real(dp)
r = real(cx)
r = real(dcx)
r = float(i)
r = sngl(dp)
dp = dble(i)
dp = dble(r)
dp = dble(cx)
dp = dble(dcx)
cx = cmplx(i)
cx = cmplx(i, j)
cx = cmplx(r)
cx = cmplx(r, s)
cx = cmplx(dp)
cx = cmplx(dp, dq)
cx = cmplx(dcx)
dcx = dcmplx(i)
dcx = dcmplx(i, j)
dcx = dcmplx(r)
dcx = dcmplx(r, s)
dcx = dcmplx(dp)
dcx = dcmplx(dp, dq)
dcx = dcmplx(cx)
i = ichar(ch)
ch = char(i)

DESCRIPTION
These functions perform conversion from one data type to another.
The function int converts to integer form its real, double precision, complex, or double complex argument. If the argument is real or double precision, int returns
the integer whose magnitude is the largest integer that does not exceed the magni-
itude of the argument and whose sign is the same as the sign of the argument
(i.e. truncation). For complex types, the above rule is applied to the real part.
ifix and idint convert only real and double precision arguments respectively.
The function real converts to real form an integer, double precision, complex, or
double complex argument. If the argument is double precision or double complex,
as much precision is kept as is possible. If the argument is one of the complex
types, the real part is returned. float and sngl convert only integer and double
precision arguments respectively.
The function dble converts any integer, real, complex, or double complex argu-
ment to double precision form. If the argument is of a complex type, the real
part is returned.
The function cmplx converts its integer, real, double precision, or double complex
argument(s) to complex form.
The function dcmplx converts to double complex form its integer, real, double pre-
cision, or complex argument(s).
Either one or two arguments may be supplied to cmplx and dcmplx. If there is
only one argument, it is taken as the real part of the complex type and an ima-
ginary part of zero is supplied. If two arguments are supplied, the first is taken
as the real part and the second as the imaginary part.
The function ichar converts from a character to an integer depending on the
character's position in the collating sequence.
The function char returns the character in the ith position in the processor col-
lating sequence where i is the supplied argument.
For a processor capable of representing n characters,
ichar(char(i)) = i for 0 ≤ i < n, and
char(ichar(ch)) = ch for any representable character ch.
NAME
getarg — return Fortran command-line argument

SYNOPSIS
character*N c
integer i

 call getarg(i, c)

DESCRIPTION
 getarg returns the i-th command-line argument of the current process. Thus, if a
 program were invoked via

 foo arg1 arg2 arg3

 getarg(2, c) would return the string "arg2" in the character variable c.

SEE ALSO
getopt(3C).
NAME
getenv — return Fortran environment variable

SYNOPSIS
character*N c
    call getenv("VARIABLE_NAME", c)

DESCRIPTION
getenv returns the character-string value of the environment variable represented by its first argument into the character variable of its second argument. If no such environment variable exists, all blanks will be returned.

SEE ALSO
getenv(3C), environ(5).
IARGC(3F) (FORTRAN Programming Language Utilities) IARGC(3F)

NAME
   iargc — return the number of command line arguments

SYNOPSIS
   integer i

   i = iargc(

DESCRIPTION
   The iargc function returns the number of command line arguments passed to the
   program. Thus, if a program were invoked via

       foo arg1 arg2 arg3

   iargc() would return 3.

SEE ALSO
   getarg(3F).
NAME
index – return location of Fortran substring

SYNOPSIS
character·N1 ch1
character·N2 ch2
integer i
i = index(ch1, ch2)

DESCRIPTION
index returns the location of substring ch2 in string ch1. The value returned is the position at which substring ch2 starts, or 0 if it is not present in string ch1. If N2 is greater than N1, a zero is returned.
NAME
len — return length of Fortran string

SYNOPSIS
character*N ch
integer i
i = len(ch)

DESCRIPTION
len returns the length of string ch.
NAME
log, alog, dlog, clog — Fortran natural logarithm intrinsic function

SYNOPSIS
real r1, r2
double precision dp1, dp2
complex cx1, cx2
r2 = alog(r1)
r2 = log(r1)
dp2 = dlog(dp1)
dp2 = log(dp1)
cx2 = clog(cx1)
cx2 = log(cx1)

DESCRIPTION
alog returns the real natural logarithm of its real argument. dlog returns the
double-precision natural logarithm of its double-precision argument. clog
returns the complex logarithm of its complex argument. The generic function
log becomes a call to alog, dlog, or clog depending on the type of its argument.

SEE ALSO
exp(3M).
NAME
\texttt{log10, alog10, dlog10} – Fortran common logarithm intrinsic function

SYNOPSIS
\begin{verbatim}
real r1, r2
double precision dp1, dp2
r2 = alog10(r1)
r2 = log10(r1)
dp2 = dlog10(dp1)
dp2 = log10(dp1)
\end{verbatim}

DESCRIPTION
\texttt{alog10} returns the real common logarithm of its real argument. \texttt{dlog10} returns
the double-precision common logarithm of its double-precision argument. The
generic function \texttt{log10} becomes a call to \texttt{alog10} or \texttt{dlog10} depending on the type
of its argument.

SEE ALSO
\texttt{exp(3M)}. 

NAME

max, max0, amax0, max1, amax1, dmax1 — Fortran maximum-value functions

SYNOPSIS

integer i, j, k, 1
real a, b, c, d
double precision dp1, dp2, dp3

l = max(i, j, k)
c = max(a, b)
dp = max(a, b, c)
k = max0(i, j)
a = amax0(i, j, k)
i = max1(a, b)
d = amax1(a, b, c)

dp3 = dmax1(dp1, dp2)

DESCRIPTION

The maximum-value functions return the largest of their arguments (of which there may be any number). max is the generic form which can be used for all data types and takes its return type from that of its arguments (which must all be of the same type). max0 returns the integer form of the maximum value of its integer arguments; amax0, the real form of its integer arguments; max1, the integer form of its real arguments; amax1, the real form of its real arguments; and dmax1, the double-precision form of its double-precision arguments.

SEE ALSO

min(3F).
MCLOCK(3F) (FORTRAN Programming Language Utilities) MCLOCK(3F)

NAME
mclock — return Fortran time accounting

SYNOPSIS

integer i
i = mclock( )

DESCRIPTION
mclock returns time accounting information about the current process and its child processes. The value returned is the sum of the current process's user time and the user and system times of all child processes.

SEE ALSO
times(2), clock(3C), system(3F).
MIL(3F) (FORTRAN Programming Language Utilities) MIL(3F)

NAME
mil: ior, iand, not, ieor, ishft, ishftc, ibits, btest, ibset, ibclr, mvbits — Fortran Military Standard functions

SYNOPSIS
integer i, k, l, m, n, len
logical b

i = ior(m, n)
i = iand(m, n)
i = not(m)
i = ieor(m, n)
i = ishft(m, k)
i = ishftc(m, k, len)
i = ibits(m, k, len)
b = btest(n, k)
i = ibset(n, k)
i = ibclr(n, k)
call mvbits(m, k, len, n, l)

DESCRIPTION
mil is the general name for the bit field manipulation intrinsic functions and subroutines from the Fortran Military Standard (MIL-STD-1753). ior, iand, not, ieor — return the same results as and, or, not, xor as defined in bool(3F).

ishft, ishftc — m specifies the integer to be shifted. k specifies the shift count. k > 0 indicates a left shift. k = 0 indicates no shift. k < 0 indicates a right shift. In ishft, zeros are shifted in. In ishftc, the rightmost len bits are shifted circularly k bits. If k is greater than the machine word-size, ishftc will not shift.

Bit fields are numbered from right to left and the rightmost bit position is zero. The length of the len field must be greater than zero.

ibits — extract a subfield of len bits from m starting with bit position k and extending left for len bits. The result field is right justified and the remaining bits are set to zero.

btest — The kth bit of argument n is tested. The value of the function is .TRUE. if the bit is a 1 and .FALSE. if the bit is 0.

ibset — the result is the value of n with the kth bit set to 1.

ibclr — the result is the value of n with the kth bit set to 0.

mvbits — len bits are moved beginning at position k of argument m to position l of argument n.

SEE ALSO
bool(3F).
NAME
    min, min0, amin0, min1, amin1, dmin1 – Fortran minimum-value functions

SYNOPSIS
    integer i, j, k, l
    real a, b, c, d
    double precision dp1, dp2, dp3
    l = min(i, j, k)
    c = min(a, b)
    dp = min(a, b, c)
    k = min0(i, j)
    a = amin0(i, j, k)
    i = min1(a, b)
    d = amin1(a, b, c)
    dp3 = dmin1(dp1, dp2)

DESCRIPTION
    The minimum-value functions return the minimum of their arguments (of which
    there may be any number). min is the generic form which can be used for all
    data types and takes its return type from that of its arguments (which must all
    be of the same type). min0 returns the integer form of the minimum value of its
    integer arguments; amin0, the real form of its integer arguments; min1, the
    integer form of its real arguments; amin1, the real form of its real arguments;
    and dmin1, the double-precision form of its double-precision arguments.

SEE ALSO
    max(3F).
NAME

mod, amod, dmod - Fortran remaindering intrinsic functions

SYNOPSIS

integer i, j, k
real r1, r2, r3
double precision dp1, dp2, dp3
k = mod(i, j)
r3 = amod(r1, r2)
r3 = mod(r1, r2)
dp3 = dmod(dp1, dp2)
dp3 = mod(dp1, dp2)

DESCRIPTION

mod returns the integer remainder of its first argument divided by its second argument. Amod and dmod return, respectively, the real and double-precision whole number remainder of the integer division of their two arguments. The generic version mod will return the data type of its arguments.
NAME
rand, irand, srand — random number generator

SYNOPSIS
integer iseed, i, irand
double precision x, rand

call srand(iseed)
i = irand( )
x = rand( )

DESCRIPTION
Irand generates successive pseudo-random integers in the range from 0 to 2**15-1. rand generates pseudo-random numbers distributed in [0, 1.0]. Srand uses its integer argument to re-initialize the seed for successive invocations of irand and rand.

SEE ALSO
rand(3C).
NAME
round: anint, dnint, nint, idnint — Fortran nearest integer functions

SYNOPSIS
integer i
real r1, r2
double precision dp1, dp2
r2 = anint(r1)
i = nint(r1)
dp2 = anint(dp1)
dp2 = dnint(dp1)
i = nint(dp1)
i = idnint(dp1)

DESCRIPTION
Anint returns the nearest whole real number to its real argument (i.e., int(a+0.5) if \( a \geq 0 \), int(a-0.5) otherwise). Dnint does the same for its double-precision argument. Nint returns the nearest integer to its real argument. Idnint is the double-precision version. Anint is the generic form of anint and dnint, performing the same operation and returning the data type of its argument. Nint is also the generic form of idnint.
NAME
sign, isign, dsign — Fortran transfer-of-sign intrinsic function

SYNOPSIS

integer i, j, k
real r1, r2, r3
double precision dp1, dp2, dp3

k = isign(i, j)
k = sign(i, j)
r3 = sign(r1, r2)
dp3 = dsign(dp1, dp2)
dp3 = sign(dp1, dp2)

DESCRIPTION
Isign returns the magnitude of its first argument with the sign of its second argument. sign and dsign are its real and double-precision counterparts, respectively. The generic version is sign and will devolve to the appropriate type depending on its arguments.
NAME
signal — specify Fortran action on receipt of a system signal

SYNOPSIS
integer i, intfc
external intfc
call signal(i, intfc)

DESCRIPTION
The argument i specifies the signal to be caught. signal allows a process to specify a function to be invoked upon receipt of a specific signal. The first argument specifies which fault or exception. The second argument specifies the function to be invoked.
NOTE: The interrupt processing function, intfc, does not take an argument.

SEE ALSO
kill(2), signal(2).
NAME
sin, dsin, csin — Fortran sine intrinsic function

SYNOPSIS
real r1, r2
double precision dp1, dp2
complex cx1, cx2
r2 = sin(r1)
dp2 = dsin(dp1)
dp2 = sin(dp1)
cx2 = csin(cx1)
cx2 = sin(cx1)

DESCRIPTION
sin returns the real sine of its real argument. Dsin returns the double-precision sine of its double-precision argument. Csin returns the complex sine of its complex argument. The generic sin function becomes dsin or csin as required by argument type.

SEE ALSO
trig(3M).
NAME
    sinh, dsinh – Fortran hyperbolic sine intrinsic function

SYNOPSIS
    real r1, r2
    double precision dp1, dp2
    r2 = sinh(r1)
    dp2 = dsinh(dp1)
    dp2 = sinh(dp1)

DESCRIPTION
    sinh returns the real hyperbolic sine of its real argument. Dsinh returns the double-precision hyperbolic sine of its double-precision argument. The generic form sinh may be used to return a double-precision value when given a double-precision argument.

SEE ALSO
    sinh(3M).
NAME
sqrt, dsqrt, csqrt — Fortran square root intrinsic function

SYNOPSIS
real r1, r2
double precision dp1, dp2
complex cx1, cx2
r2 = sqrt(r1)
dp2 = dsqrt(dp1)
dp2 = sqrt(dp1)
cx2 = csqrt(cx1)
cx2 = sqrt(cx1)

DESCRIPTION
sqrt returns the real square root of its real argument. Dsqrt returns the double-
precision square root of its double-precision argument. Csqrt returns the com-
plex square root of its complex argument. sqrt, the generic form, will become
dsqrt or csqrt as required by its argument type.

SEE ALSO
exp(3M).
NAME

strcmp: lge, lgt, lle, llt — string comparison intrinsic functions

SYNOPSIS

character*N a1, a2
logical l

1 = lge(a1, a2)
1 = lgt(a1, a2)
1 = lle(a1, a2)
1 = llt(a1, a2)

DESCRIPTION

These functions return .TRUE. if the inequality holds and .FALSE. otherwise.
NAME
 system — issue a shell command from Fortran

SYNOPSIS
 character*N c
 call system(c)

DESCRIPTION
 system causes its character argument to be given to sh(1) as input, as if the string
 had been typed at a terminal. The current process waits until the shell has com-
 pleted.

SEE ALSO
 exec(2), system(3S).
NAME
  tan, dtan – Fortran tangent intrinsic function

SYNOPSIS
  real r1, r2
  double precision dp1, dp2
  r2 = tan(r1)
  dp2 = dtan(dp1)
  dp2 = tan(dp1)

DESCRIPTION
  \textit{tan} returns the real tangent of its real argument. \textit{Dt}an returns the double-
  precision tangent of its double-precision argument. The generic \textit{tan} function
  becomes \textit{dtan} as required with a double-precision argument.

SEE ALSO
  trig(3M).
NAME
tanh, dtanh — Fortran hyperbolic tangent intrinsic function

SYNOPSIS
real r1, r2
double precision dp1, dp2
r2 = tanh(r1)
dp2 = dtanh(dp1)
dp2 = tanh(dp1)

DESCRIPTION
tanh returns the real hyperbolic tangent of its real argument. Dtanh returns the
double-precision hyperbolic tangent of its double-precision argument. The gen­
eric form tanh may be used to return a double-precision value given a double-
precision argument.

SEE ALSO
sinh(3M).
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. Usually, the header files containing these structure declarations can be found in the directories /usr/include or /usr/include/sys. For inclusion in C language programs, however, the syntax #include <filename.h> or #include <sys/filename.h> should be used.
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 will make 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 UNIX system header (if the file is link editor output), a table of section headers, relocation information, (optional) line numbers, a symbol table, and a string table. The order is given below.

File header.
UNIX 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 will be 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 the 3B2 computer the text segment starts at location 0x80800000.

The a.out file produced by ld(1) has the magic number 0413 in the first field of the UNIX system header. The headers (file header, UNIX 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
will equal 0x80800000 plus the size of the headers, and will vary depending upon the number of section headers in the a.out file. In an a.out file with three sections (.text, .data, and .bss), the first text address is at 0x808000A8 on the 3B2 computer. The text segment is not writable by the program; if other processes are executing the same a.out file, the processes will share a single text segment.

The data segment starts at the next 512K boundary past the last text address. The first data address is determined by the following: If an a.out file were split into 8K chunks, one of the chunks would contain both the end of text and the beginning of data. When the core image is created, that chunk will appear twice; once at the end of text and once at the beginning of data (with some unused space in between). The duplicated chunk 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 8K. If the last text address is a multiple of 8K no duplication is necessary.

On the 3B2 computer the stack begins at location 0xC0020000 and grows toward higher 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 will appear in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, there will be a relocation entry for the word, the storage class of the symbol-table entry for the symbol will be marked as an “external symbol”, and the value and section number of the symbol-table entry will be undefined. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

**File Header**

The format of the filehdr header is

```c
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 */
};
```

**UNIX System Header**

The format of the UNIX system header is

```c
typedef struct aouthdr
{
  short     magic;        /* magic number */
```
A.OUT(4)

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 */
}

Section Header

The format of the section header is

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_relpnr; /* file ptr to relocation */
    long s_lnpnr; /* file ptr to line numbers */
    unsigned short s_nrelnc; /* # reloc entries */
    unsigned short s_nlnno; /* # line number entries */
    long sflags; /* flags */
};

Relocation

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be 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_relpnr from the section header. If there is no relocation information, s_relpnr is 0.
Symbol Table

The format of each symbol in the symbol table is

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

struct syment
{
    union
    {
        char  _n_name[SYMNMLEN]; /* name of symbol */
    } _n;
    long _n_zeros; /* == 0L if in string table */
    long _n_offset; /* location in string table */
    char *_n_nptr[2]; /* allows overlaying */
    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 */
};
```

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 short x_lno;
                unsigned short x_size;
            } x_lnsz;
            long x_fsize;
        } x_misc;
        union {
            struct {
                long x_Lnopptr;
                long x_endndx;
            } x_fcn;
            struct {
                unsigned short x_dimen[DIMNUM];
            } x_ary;
        } x_sym;
    }
    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 */
    ushort ac_uid;
    ushort ac_gid;
    dev_t ac_tty;
    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 transfrd 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 super-user privileges */
#define ACCTF 0300 /* record type: 00 = acct */

In ac_flag, the AFORK flag is turned on by each fork(2) and turned off by an exec(2). The ac_comm field is inherited from the parent process and is reset by any exec. Each time the system charges the process with a clock tick, it also adds to ac_mem the current process size, computed as follows:

\[(\text{data size}) + (\text{text size}) / (\text{number of in-core processes using text})\]

The value of ac_mem/(ac_stime + ac_utime) can be viewed as an approximation to the mean process size, as modified by text-sharing.
The structure `tacct.h`, which resides with the source files of the accounting commands, represents the total accounting format used by the various accounting commands:

```c
/*
 * 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).

BUGS
The `ac_mem` value for a short-lived command gives little information about the actual size of the command, because `ac_mem` may be incremented while a different command (e.g., the shell) is being executed by the process.
NAME
ar — common archive file format

SYNOPSIS
#include <ar.h>

DESCRIPTION
The archive command ar(1) is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor ld(1).

Each archive begins with the archive magic string.

#define ARMAG "!<arch>
/* magic string */
#define SARMAG 8 /* length of magic string */

Each archive which contains common object files [see a.out(4)] includes an archive symbol table. This symbol table is used by the link editor ld(1) to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and/or updated by ar.

Following the archive magic string are the archive file members. Each file member is preceded by a file member header which is of the following format:

#define ARFMAG "\n" /* header trailer string */

struct ar_hdr /* file member header */ {
  char ar_name[16]; /* '/' terminated file member name */
  char ar_date[12]; /* file member date */
  char ar_uid[6]; /* file member user identification */
  char ar_gid[6]; /* file member group identification */
  char ar_mode[8]; /* file member mode (octal) */
  char ar_size[10]; /* file member size */
  char ar_fmag[2]; /* header trailer string */
};

All information in the file member headers is in printable ASCII. The numeric information contained in the headers is stored as decimal numbers (except for ar_mode which is in octal). Thus, if the archive contains printable files, the archive itself is printable.

The ar_name field is blank-padded and slash (/) terminated. The ar_date field is the modification date of the file at the time of its insertion into the archive. Common format archives can be moved from system to system as long as the portable archive command ar(1) is used. Conversion tools such as convert(1) exist to aid in the transportation of non-common format archives to this format.

Each archive file member begins on an even byte boundary; a newline is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.
Notice there is no provision for empty areas in an archive file.
If the archive symbol table exists, the first file in the archive has a zero length name (i.e., \texttt{ar\_name[0] == "/"}). The contents of this file are as follows:

- The number of symbols. Length: 4 bytes.
- The array of offsets into the archive file. Length: 4 bytes • “the number of symbols”.
- The name string table. Length: $ar\_size - (4 \times ("the number of symbols" + 1)).$

The number of symbols and the array of offsets are managed with \texttt{sgetl} and \texttt{sputl}. The string table contains exactly as many null terminated strings as there are elements in the offsets array. Each offset from the array is associated with the corresponding name from the string table (in order). The names in the string table are all the defined global symbols found in the common object files in the archive. Each offset is the location of the archive header for the associated symbol.

SEE ALSO
\texttt{ar(1)}, \texttt{ld(1)}, \texttt{strip(1)}, \texttt{sputl(3X)}, \texttt{a.out(4)}.

WARNINGS
\texttt{Strip(1)} will remove all archive symbol entries from the header. The archive symbol entries must be restored via the \texttt{ts} option of the \texttt{ar(1)} command before the archive can be used with the link editor \texttt{ld(1)}.
NAME
checklist — list of file systems processed by fsck and ncheck

DESCRIPTION
checklist resides in directory /etc and contains a list of, at most, 15 special file names. Each special file name is contained on a separate line and corresponds to a file system. Each file system will then be automatically processed by the fsck(1M) command.

FILES
/etc/checklist

SEE ALSO
fsck(1M), ncheck(1M) in the System Administrator's Reference Manual.
NAME

core — format of core image file

DESCRIPTION

The UNIX system writes out a core image of a terminated process when any of various errors occur. See signal(2) for the list of reasons; the most common 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 it can be; normal access controls apply). A process with an effective user ID different from the real user ID will not produce a core image.

The first section of the core image is a copy of the system’s per-user data for the process, including the registers as they were at the time of the fault. The size of this section depends on the parameter usize, which is defined in <sys/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>. Not included in this file are the locations of the registers. These are outlined in <sys/reg.h>.

SEE ALSO

sdb(1), setuid(2), signal(2).
NAME
cpio — format of cpio archive

DESCRIPTION
The header structure, when the -c option of cpio(1) is not used, is:

```c
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;
```

When the -c option is used, the header information is described by:

```c
sscanf(Chdr,"%60%60%60%60%60%60%60%60%11lo%60%11lo%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 path name 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).
NAME
dir — format of directories

SYNOPSIS
#include <sys/dir.h>

DESCRIPTION
A directory behaves exactly like an ordinary file, save 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:

    #ifndef  DIRSIZ
    #define  DIRSIZ 14
    #endif

    struct  direct
    {
        ushort  d_ino;
        char    d_name[DIRSIZ];
    };

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; there is
no parent, so .. has the same meaning as ..

SEE ALSO
    fs(4).
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 differ-
et file system types. A set of these structures is returned by the getdents(2) system call.

The dirent structure is defined below.
struct dirent {
    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 will always result in the
next entry being on a long boundary.

FILES
/usr/include/sys/dirent.h

SEE ALSO
getdents(2).
.EDT_SWAPP(4)

NAME
.edt_swapp — software application file

DESCRIPTION
The .edt_swapp file is read by filledt(8) on its second pass during the self-
configuration process to rename specified Equipped Device Table (EDT) entries.
The file has the following format:

<table>
<thead>
<tr>
<th>SLOT</th>
<th>SWNAME</th>
<th>HWNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>FF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The number in the slot field specifies the entry in the EDT to be updated. The
SWNAME column contains the new name which will be associated with this
board. The HWNAME field contains the name which the board was associated
with generically. The last line contains 'FF' for the slot number to signal the end
of the data to the firmware. This file can be displayed by the editsa -l command.

WARNINGS
This file is not to be edited directly. Updates to it must be done through editsa.

SEE ALSO
NAME

filehdr — file header for common object files

SYNOPSIS

```c
#include <filehdr.h>
```

DESCRIPTION

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

```c
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_nsym; /* # 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 UNIX system optional header is 28-bytes. The valid magic numbers are given below:

```c
#define FBOMAGIC 0560 /* 3B2 and 3B5 computers */
#define N3BMAGIC 0550 /* 3B20 computer */
#define NTVMAGIC 0551 /* 3B20 computer */
#define VAXWRMAGIC 0570 /* VAX writable text segments */
#define VAXROMAGIC 0575 /* VAX read only sharable text segments */
```

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

```c
#define F_RELFLG 0000001 /* relocation entries stripped */
#define F_EXEC 0000002 /* file is executable */
#define F_LNNO 0000004 /* line numbers stripped */
#define F_LSyms 0000010 /* local symbols stripped */
#define F_MINIMAL 0000020 /* minimal object file */
#define F_UPDATE 0000040 /* update file, ogen produced */
#define F_SWABD 0000100 /* file is "pre-swabbed" */
#define F_AR16WR 0000200 /* 16-bit DEC host */
#define F_AR32WR 0000400 /* 32-bit DEC host */
#define F_AR32W 0001000 /* non-DEC host */
#define F_PATCH 0002000 /* "patch" list in opt hdr */
#define F_BM32ID 0160000 /* WE32000 family ID field */
#define F_BM32B 0020000 /* file contains WE 32100 code */
#define F_BM32MAU 0004000 /* file reqs MAU to execute */
#define F_BM32RST 0001000 /* this object file contains restore work around [3B5/3B2 only] */
```
SEE ALSO
   time(2), fseek(3S), a.out(4).
NAME
fs: file system — format of system volume

SYNOPSIS
#include <sys/filsys.h>
#include <sys/types.h>
#include <sys/param.h>

DESCRIPTION
Every file system storage volume has a common format for certain vital information. Every such volume is divided into a certain number of 512-byte long sectors. Sector 0 is unused and is available to contain a bootstrap program or other information.

Sector 1 is the super-block. The format of a super-block is:

```c
struct filsys {
    ushort s_isize; /* size in blocks of i-list */
    daddr_t s_fsize; /* size in blocks of entire volume */
    short s_nfree; /* number of addresses in s_free */
    daddr_t s_free[NICFREE]; /* free block list */
    short s_ninode; /* number of i-nodes in s_inode */
    ushort s_inode[NICINOD]; /* free i-node list */
    char s_flock; /* lock during free list manipulation */
    char s_ilock; /* lock during i-list manipulation */
    char s_fmod; /* super block modified flag */
    char s_ronly; /* mounted read-only flag */
    time_t s_time; /* last super block update */
    short s_dinfo[4]; /* device information */
    daddr_t s_tfree; /* total free blocks */
    ushort s_tinode; /* total free i-nodes */
    char s_fname[6]; /* file system name */
    char s_fpack[6]; /* file system pack name */
    long s_fill[12]; /* ADJUST to make sizeof filsys be 512 */
    long s_state; /* file system state */
    long s_magic; /* magic number to denote new file system */
    long s_type; /* type of new file system */
};
```

#define FsMAGIC 0xfd187e20 /* s_magic number */
#define Fs1b 1 /* 512-byte block */
#define Fs2b 2 /* 1024-byte block */
#define FsOKAY 0x7c269d38 /* s_state: clean */
#define FsACTIVE 0x5e72d81a /* s_state: active */
#define FsBAD 0xcb096f43 /* s_state: bad root */
#define FsBADBLK 0xbadbc14b /* s_state: bad block corrupted it */
S_tree indicates the file system type. Currently, two types of file systems are supported: the original 512-byte logical block and the improved 1024-byte logical block. S_magic is used to distinguish the original 512-byte oriented file systems from the newer file systems. If this field is not equal to the magic number, fsMAGIC, the type is assumed to be fs1b, otherwise the s_type field is used. In the following description, a block is then determined by the type. For the original 512-byte oriented file system, a block is 512-bytes. For the 1024-byte oriented file system, a block is 1024-bytes or two sectors. The operating system takes care of all conversions from logical block numbers to physical sector numbers.

S_state indicates the state of the file system. A cleanly unmounted, not damaged file system is indicated by the FsOKAY state. After a file system has been mounted for update, the state changes to FsACTIVE. A special case is used for the root file system. If the root file system appears damaged at boot time, it is mounted but marked FsBAD. Lastly, after a file system has been unmounted, the state reverts to FsOKAY.

S_isize is the address of the first data block after the i-list; the i-list starts just after the super-block, namely in block 2; thus the i-list is s_isize−2 blocks long. S_fsize is the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block numbers; if an "impossible" block number is allocated from the free list or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The s_free array contains, in s_free[1], ..., s_free[s_nfree−1], up to 49 numbers of free blocks. S_free[0] is the block number of the head of a chain of blocks constituting the free list. The first long in each free-chain block is the number (up to 50) of free-block numbers listed in the next 50 longs of this chain member. The first of these 50 blocks is the link to the next member of the chain. To allocate a block: decrement s_nfree, and the new block is s_free[s_nfree]. If the new block number is 0, there are no blocks left, so give an error. If s_nfree became 0, read in the block named by the new block number, replace s_nfree by its first word, and copy the block numbers in the next 50 longs into the s_free array. To free a block, check if s_nfree is 50; if so, copy s_nfree and the s_free array into it, write it out, and set s_nfree to 0. In any event set s_free[s_nfree] to the freed block’s number and increment s_nfree.

S_tfree is the total free blocks available in the file system.

S_ninode is the number of free i-numbers in the s_inode array. To allocate an i-node: if s_ninode is greater than 0, decrement it and return s_inode[s_ninode]. If it was 0, read the i-list and place the numbers of all free i-nodes (up to 100) into the s_inode array, then try again. To free an i-node, provided s_ninode is less than 100, place its number into s_inode[s_ninode] and increment s_ninode. If s_ninode is already 100, do not bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the i-node is really free or not is maintained in the i-node itself.

S_tinode is the total free i-nodes available in the file system.
S_flock and s_ilock are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of s_fmod on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

S_ronly is a read-only flag to indicate write-protection.

S_time is the last time the super-block of the file system was changed, and is the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (GMT). During a reboot, the s_time of the super-block for the root file system is used to set the system's idea of the time.

S_fname is the name of the file system and s_fpack is the name of the pack.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. Also, i-nodes are 64 bytes long. I-node 1 is reserved for future use. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. For the format of an i-node and its flags, see inode(4).

SEE ALSO
mount(2), inode(4).
fsck(1M), fsdb(1M), mkfs(1M) in the System Administrator's Reference Manual.
NAME
fspec — format specification in text files

DESCRIPTION
It is sometimes convenient to maintain text files on the UNIX system with non-
standard tabs, (i.e., tabs which 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
UNIX 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:

ttabs  The t parameter specifies the tab settings for the file. The value of
tabs must be one of the following:
   1. a list of column numbers separated by commas, indicating tabs
      set at the specified columns;
   2. a - followed immediately by an integer n, indicating tabs at
      intervals of n columns;
   3. a - followed by the name of a "canned" tab specification.

Standard tabs are specified by t−8, or equivalently, t1,9,17,25, etc.
The canned tabs which are recognized are defined by the tabs(1)
command.

ssize  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.

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

d  The d parameter takes no value. Its presence indicates that the line
containing the format specification is to be deleted from the con-
verted file.

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

Default values, 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 \texttt{d} parameter.

\textbf{SEE ALSO}
\\texttt{ed(1)}, \texttt{newform(1)}, \texttt{tabs(1)} in the \textit{User's Reference Manual}.
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:

<table>
<thead>
<tr>
<th>column 1</th>
<th>block special file name of file system or advertised remote resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>column 2</td>
<td>mount-point directory</td>
</tr>
<tr>
<td>column 3</td>
<td>&quot;-r&quot; if to be mounted read-only; &quot;-d[r]&quot; if remote</td>
</tr>
<tr>
<td>column 4</td>
<td>(optional) file system type string</td>
</tr>
<tr>
<td>column 5+</td>
<td>ignored</td>
</tr>
</tbody>
</table>

White-space separates columns. Lines beginning with "# " are comments. Empty lines are ignored.

A file-system-table might read:

```
/dev/dsk/c1d0s2 /usr S51K
/dev/dsk/c1d1s2 /usr/src -r
adv_resource /mnt -d
```

FILES
/etc/fstab

SEE ALSO
mount(1M), mountall(1M), rmountall(1M) in the System Administrator's Reference Manual.
NAME
gettydefs — speed and terminal settings used by getty

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

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

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

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

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

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

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

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

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

If getty is called without a second argument, then the first entry of /etc/gettydefs is used, thus making the first entry of /etc/gettydefs the default entry. It is also used if getty can not find the specified label. If /etc/gettydefs itself is missing, there is one entry built into the command which will bring up a terminal at 300 baud.
It is strongly recommended that after making or modifying /etc/gettydefs, it be run through getty with the check option to be sure there are no errors.

FILES
/etc/gettydefs

SEE ALSO
ioctl(2).
NAME

gps — graphical primitive string, format of graphical files

DESCRIPTION

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.

A GPS is composed of five types of graphical data or primitives.

GPS PRIMITIVES

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 seg­
ments 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 a
circular arc through the points is drawn; and if more than three, 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 begin­
ing 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 PARAMETERS

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.

ORGANIZATION

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  Ssw 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.

SEE ALSO

graphics(1G), stat(1G), toc(1G) in the User's Reference Manual.
GROUP(4) group — group file

DESCRIPTION

group contains for each group the following information:

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

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

FILES

/etc/group

SEE ALSO

passwd(4).
NAME

`inittab` — script for the init process

DESCRIPTION

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

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

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

Each entry is delimited by a newline, however, a backslash (`\`) preceding a newline indicates a continuation of the entry. Up to 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:

- **id**: This is one or two characters used to uniquely identify an entry.
- **rstate**: This defines 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 by a number ranging from 0 through 6. 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` will be 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`. 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). They differ from `run-levels` in that `init` can never enter `run-level` a, b or c. Also, a request for the execution of any of these processes does not change the current `run-level`. Furthermore, a process started by an a, b or c command is not killed when `init` changes levels. 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 then 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 then 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** will 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 will not be 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 will run 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 see **signal**(2)].

powerwait Execute the process associated with this entry only when **init** receives a power fail signal (SIGPWR) and wait until it terminates before continuing any processing of **inittab**.

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

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

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

**sysinit**

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

### process

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

**FILES**

\texttt{/etc/inittab}

**SEE ALSO**

\texttt{exec(2), open(2), signal(2).}\n\texttt{getty(1M), init(1M)} in the \textit{System Administrator's Reference Manual}.\n\texttt{sh(1), who(1)} in the \textit{User's Reference Manual}.
NAME
inode — format of an i-node

SYNOPSIS
#include <sys/types.h>
#include <sys/in.h>

DESCRIPTION
An i-node for a plain file or directory in a file system has the following structure
defined by <sys/in.h>.

/* Inode structure as it appears on a disk block. */
struct dinode
{
    ushort di_mode; /* mode and type of file */
    short di_link;  /* number of links to file */
    ushort di_uid;   /* owner’s user id */
    ushort di_gid;   /* owner’s group id */
    off_t di_size;  /* number of bytes in file */
    char  di_addr[40]; /* disk block addresses */
    time_t di_atime; /* time last accessed */
    time_t di_mtime; /* time last modified */
    time_t di_ctime; /* time of last file status change */
};

/*
* the 40 address bytes:
* 39 used; 13 addresses
* of 3 bytes each.
*/

For the meaning of the defined types off_t and time_t see types(5).

SEE ALSO
stat(2), fs(4), types(5).
NAME
issue — issue identification file

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

FILES
/etc/issue

SEE ALSO
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 effectively 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 Idfile, 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 ldopen[see ldopen(3X)]
open a common object file

ldclose(3X) and ldclose[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

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

ldshread(3X) and ldshread[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

ldseek(3X) and ldnseek[see ldseek(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 ldtnindex(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), ldtnindex(3X) return either SUCCESS or FAILURE, both constants defined in ldfcn.h. ldopen(3X) and ldaopen[see ldopen(3X)] both 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), ldhdrread(3X), ldllread(3X), ldlseek(3X), ldohseek(3X), ldopen(3X), ldread(3X), ldshread(3X), ldsseek(3X), ldttbindex(3X), ldttbread(3X), ldttbseek(3X), stdio(3S), intro(5).

WARNING
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 ARG_MAX 5120 /* max length of arguments to exec */
#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 CHILD_MAX 25 /* max # of processes per user id */
#define CLK_TCK 100 /* # of clock ticks per second */
#define DBL_DIG 53 /* digits of precision of a "double" */
#define DBL_MAX 1.79769313486231470e+308 /* max decimal value of a "double" */
#define DBL_MIN 2.2250738585072014e-308 /* min decimal value of a "double" */
#define FCHR_MAX 127 /* max length of arguments to exec */
#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 LINK_MAX 256 /* 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 NAME_MAX 14 /* max # of characters in a file name */
#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 256 /* max # of characters in a path name */
#define PID_MAX 30000 /* max value for a process ID */
#define PIPE_BUF 512 /* max bytes atomic in write to a pipe */
#define PIPE_MAX 512 /* 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 1024 /* # bytes in a physical I/O block */
#define SYS_NMLN 9 /* # of chars in uname-returned strings */
#define UID_MAX 4294967296 /* max decimal value of an "unsigned" */
#define USI_MAX 2147483647 /* max decimal value of a "short" */
#define WORD_BIT 2 /* # of bits in a "word" or "int" */
NAME
  linenum — line number entries in a common object file

SYNOPSIS
  #include <linenum.h>

DESCRIPTION
  The cc command generates an entry in the object file for each C source line on
  which a breakpoint is possible [when invoked with the -g option; see cc(1)].
  Users can then reference line numbers when using the appropriate software test
  system [see sdb(1)]. The structure of these line number entries appears below.

  struct lineno
  {
    union
    {
      long    l_symndx;
      long    l_paddr;
      l_addr;
    }
    unsigned short l_inno;
  };

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

  l_addr      l_inno
  function symtab index  0
  physical address       line
  physical address       line
  ...
  function symtab index  0
  physical address       line
  physical address       line
  ...

SEE ALSO
  cc(1), sdb(1), a.out(4).
NAME

master — master configuration database

DESCRIPTION

The master configuration database is a collection of files. Each file contains configuration information for a device or module that may be included in the system. A file is named with the module name to which it applies. This collection of files is maintained in a directory called /etc/master.d. Each individual file has an identical format. For convenience, this collection of files will be referred to as the master file, as though it was a single file. This will allow a reference to the master file to be understood to mean the individual file in the master.d directory that corresponds to the name of a device or module. The file is used by the mkboot(1M) program to obtain device information to generate the device driver and configurable module files. It is also used by the sysdef(1M) program to obtain the names of supported devices. master consists of two parts; they are separated by a line with a dollar sign ($) in column 1. Part 1 contains device information for both hardware and software devices, and loadable modules. Part 2 contains parameter declarations used in part 1. Any line with an asterisk (*) in column 1 is treated as a comment.

Part 1, Description

Hardware devices, software drivers and loadable modules are defined with a line containing the following information. Field 1 must begin in the left most position on the line. Fields are separated by white space (tab or blank).

Field 1: element characteristics:
   o specify only once
   r required device
   b block device
   c character device
   a generate segment descriptor array
   t initialize cdevsw[].d_ttys
   s software driver
   f STREAMS driver
   m STREAMS module
   x not a driver; a loadable module

number The first interrupt vector for an integral device
   "-" if none.

Field 2: number of interrupt vectors required by a hardware device:
   "-" if none.

Field 3: handler prefix (4 chars. maximum)

Field 4: software driver external major number; "-" if not a software driver, or to be assigned during execution of drvinstall(1M)

Field 5: number of sub-devices per device; "-" if none

Field 6: interrupt priority level of the device; "-" if none

Field 7: dependency list (optional); this is a comma separated list of other drivers or modules that must be present in the configuration if this module is to be included

For each module, two classes of information are required by mkboot(1M): external routine references and variable definitions. Routine and
variable definition lines begin with white space and immediately follow
the initial module specification line. These lines are free form, thus they
may be continued arbitrarily between non-blank tokens as long as the
first character of a line is white space.

Part 1, Routine Reference Lines
If the UNIX system kernel or other dependent module contains external refer­
ences to a module, but the module is not configured, then these external refer­
ences would be undefined. Therefore, the routine reference lines are used to pro­
vide the information necessary to generate appropriate dummy functions at boot
time when the driver is not loaded.
Routine references are defined as follows:

Field 1: routine name ()
Field 2: the routine type: one of
{-nosys} routine_name() {return nosys();}
{-nodev} routine_name() {return nodev();}
{-false} routine_name() {return 0;}
{-true} routine_name() {return 1;}

Part 1, Variable Definition Lines
Variable definition lines are used to generate all variables required by the module.
The variable generated may be of arbitrary size, be initialized or not, or be
arrays containing an arbitrary number of elements.
Variable references are defined as follows:

Field 1: variable name
Field 2: [ expr ] — optional field used to indicate array size
Field 3: (length) — required field indicating the size of the variable
Field 4: ={ expr,... } — optional field used to initialize individual ele­
ments of a variable

The length field is mandatory. It is an arbitrary sequence of length specifiers,
each of which may be one of the following:

%-i an integer
%-l a long integer
%-s a short integer
%-c a single character
%-number a field which is number bytes long
%-number c a character string which is number bytes long

For example, the length field

( %-8c %-l %0x58 %-l %-c %-c )

could be used to identify a variable consisting of a character string 8-bytes long,
a long integer, a 0x58 byte structure of any type, another long integer, and two
characters. Appropriate alignment of each % specification is performed
(\%number is word aligned) and the variable length is rounded up to the next word boundary during processing.
The expressions for the optional array size and initialization are infix expressions consisting of the usual operators for addition, subtraction, multiplication, and division: +, −, *, and /. Multiplication and division have the higher precedence, but parentheses may be used to override the default order. The built-in functions min and max accept a pair of expressions, and return the appropriate value. The operands of the expression may be any mixture of the following:

&name address of name where name is any symbol defined by the kernel, any module loaded or any variable definition line of any module loaded

#name sizeof name where name is any variable name defined by a variable definition for any module loaded; the size is that of the individual variable—not the size of an entire array

#C number of controllers present; this number is determined by the EDT for hardware devices, or by the number provided in the system file for non-hardware drivers or modules

#C(name) number of controllers present for the module name; this number is determined by the EDT for hardware devices, or by the number provided in the system file for non-hardware drivers or modules

#D number of devices per controller taken directly from the current master file entry

#D(name) number of devices per controller taken directly from the master file entry for the module name

#M the internal major number assigned to the current module if it is a device driver; zero of this module is not a device driver

#M(name) the internal major number assigned to the module name if it is a device driver: zero if that module is not a device driver

name value of a parameter as defined in the second part of master

number arbitrary number (octal, decimal, or hex allowed)

string a character string enclosed within double quotes (all of the character string conventions supported by the C language are allowed); this operand has a value which is the address of a character array containing the specified string

When initializing a variable, one initialization expression should be provided for each \%i, \%1, \%s, or \%c of the length field. The only initializers allowed for a \%number c are either a character string (the string may not be longer than number), or an explicit zero. Initialization expressions must be separated by commas, and variable initialization will proceed element by element. Note that \%number specification cannot be initialized—they are set to zero. Only the first element of an array can be initialized, the other elements are set to zero. If there are more initializers than size specifications, it is an error and execution of
the *mkboot*(1M) program will be aborted. If there are fewer initializations than size specifications, zeros will be used to pad the variable. For example:

```
={ "V2.L1", #C*#D, max(10, #D), #C(OTHER), #M(OTHER) }
```

would be a possible initialization of the variable whose length field was given in the preceding example.

### Part 2, Description

*Parameter* declarations may be used to define a value symbolically. Values can be associated with identifiers and these identifiers may be used in the *variable definition* lines.

Parameters are defined as follows:

- Field 1: identifier (8 characters maximum)
- Field 2: =
- Field 3: value, the value may be a number (decimal, octal, or hex allowed), or a string

#### EXAMPLE

A sample *master* file for a tty device driver would be named "atty" if the device appeared in the EDT as "ATTY". The driver is a character device, the driver prefix is at, two interrupt vectors are used, and the interrupt priority is 6. In addition, another driver named "ATLOG" is necessary for the correct operation of the software associated with this device.

```
*FLAG #VEC PREFIX SOFT #DEV IPL DEPENDENCIES/VARIABLES
 tca 2 at  2 6  ATLOG
     atpoint(){false}
at_tty[#C*#D] (%0x58)
at_cnt(%i) ={ #C*#D }
at_logmaj(%i) ={ #M(ATLOG) }
at_id(%8c) ={ ATID }
at_table(%i%1%31%s)
    ={ max(#C,ATMAX),
      &at_tty,
      #C }
```

ATID = "fred"
ATMAX = 6

This *master* file will cause a routine named *atpoint* to be generated by the boot program if the ATTY driver is not loaded, and there is a reference to this routine from any other module loaded. When the driver is loaded, the variables at_tty, at_cnt, at_logmaj, at_id, and at_table will be allocated and initialized as specified. Due to the t flag, the d_ttys field in the character device switch table will be initialized to point to at_tty (the first variable definition line contains the variable whose address will be stored in d_ttys). The ATTY driver would reference these variables by coding:
extern struct tty at tty[];
extern int at_cnt;
extern int at_logmaj;
extern char at_id[8];
extern struct {
    int member1;
    struct tty *member2;
    char junk[31];
    short member3;
} at_table;

FILES

/etc/master.d/*

SEE ALSO

system(4),
drvinstall(1M), mkboot(1M), sysdef(1M) in the System Administrator's Reference Manual.
NAME
mnttab — mounted file system table

SYNOPSIS
#include <mnttab.h>

DESCRIPTION
mnttab resides in directory /etc and contains a table of devices, mounted by the
mount(1M) command, in the following structure as defined by <mnttab.h>:

```c
struct mnttab {
    char    mt_dev[32];
    char    mt_filsys[32];
    short   mt_ro_flg;
    time_t  mt_time;
};
```

Each entry is 70 bytes in length; the first 32 bytes are the null-padded name of
the place where the special file is mounted; the next 32 bytes represent the null-
padded root name of the mounted special file; the remaining 6 bytes contain the
mounted special file's read/write permissions and the date on which it was
mounted.

The maximum number of entries in mnttab is based on the system parameter
N MOUNT located in /etc/master.d/kernel, which defines the number of allow-
able mounted special files.

SEE ALSO
NAME
passwd — password file

DESCRIPTION
passwd contains for each user the following information:

   login name
   encrypted password
   numerical user ID
   numerical group ID
   GCOS job number, box number, optional GCOS user ID
   initial working directory
   program to use as shell

This is an ASCII file. Each field within each user’s entry is separated from the
next by a colon. The GCOS field is used only when communicating with that
system, and in other installations can contain any desired information. Each
user is separated from the next by a new-line. If the password field is null, no
password is demanded; if the shell field is null, the shell itself is used.

This file resides in directory /etc. Because of the encrypted passwords, it can
and does have general read permission and can be used, for example, to map
numerical user IDs to names.

The encrypted password consists of 13 characters chosen from a 64-character
alphabet (., /, 0–9, A–Z, a–z), except when the password is null, in which case
the encrypted password is also null. Password aging is effected for a particular
user if his encrypted password in the password file is followed by a comma and
a non-null string of characters from the above alphabet. (Such a string must be
introduced in the first instance by the super-user.)

The first character of the age, M say, denotes the maximum number of weeks for
which a password is valid. A user who attempts to login after his password has
expired will be forced to supply a new one. The next character, m say, denotes
the minimum period in weeks which must expire before the password may be
changed. The remaining characters define the week (counted from the begin­
ing 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 will be forced to change his pass­
word the next time he logs in (and the “age” will disappear from his entry in
the password file). If m > M (signified, e.g., by the string ./) only the super-
user will be able to change the password.

FILES
/etc/passwd

SEE ALSO
a64l(3C), getpwent(3C), group(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 I, 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". 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
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 plot(3X) code in a different source file than the curses(3X) code,
and/or #undef move() and erase() in the *plot*(3X) code.
NAME

pnch — file format for card images

DESCRIPTION

The PNCH format is a convenient representation for files consisting of card images in an arbitrary code.

A PNCH file is a simple concatenation of card records. A card record consists of a single control byte followed by a variable number of data bytes. The control byte specifies the number (which must lie in the range 0-80) of data bytes that follow. The data bytes are 8-bit codes that constitute the card image. If there are fewer than 80 data bytes, it is understood that the remainder of the card image consists of trailing blanks.
NAME
profile — setting up an environment at login time

SYNOPSIS
/etc/profile
$HOME/.profile

DESCRIPTION
All users who have the shell, sh(1), as their login command have the commands in these files executed as part of their 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 line

. /etc/TIMEZONE

included early in /etc/profile (see timezone(4)).

The file $HOME/.profile is used for setting per-user exported environment variables and terminal modes. The following example is typical (except for the comments):

# 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:
done
#
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

tput init
#
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' echo
FILES

/etc/TIMEZONE timezone environment
$HOME/.profile user-specific environment
/etc/profile system-wide environment

SEE ALSO

terminfo(4), timezone(4), environ(5), term(5),
env(1), login(1), mail(1), sh(1), stty(1), su(1), tput(1) in the *User's Reference Manual*.

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

User's Guide.

Chapter 10 in the *Programmer's Guide*.

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
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 will be in the following format.

```c
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
#define R_DIR32 06
#define R_DIR32S 012

As the link editor reads each input section and performs relocation, the relocation entries are read. They direct how references found within the input section are treated.

R_ABS The reference is absolute and no relocation is necessary. The entry will be ignored.
R_DIR32 A direct 32-bit reference to the symbol’s virtual address.
R_DIR32S A direct 32-bit reference to the symbol’s virtual address, with the 32-bit value stored in the reverse order in the object file.

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 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
rfmaster — Remote File Sharing name server master file

DESCRIPTION
The rfmaster file is an ASCII file that identifies the hosts that are responsible for providing primary and secondary domain name service for Remote File Sharing domains. This file contains a series of records, each terminated by a newline; a record may be extended over more than one line by escaping the newline character with a backslash ("\"). The fields in each record are separated by one or more tabs or spaces. Each record has three fields:

name type data

The type field, which defines the meaning of the name and data fields, has three possible values:

- **p** The p type defines the primary domain name server. For this type, name is the domain name and data is the full host name of the machine that is the primary name server. The full host name is specified as domain.nodename. There can be only one primary name server per domain.

- **s** The s type defines a secondary name server for a domain. Name and data are the same as for the p type. The order of the s entries in the rfmaster file determines the order in which secondary name servers take over when the current domain name server fails.

- **a** The a type defines a network address for a machine. Name is the full domain name for the machine and data is the network address of the machine. The network address can be in plain ASCII text or it can be preceded by a \x to be interpreted as hexadecimal notation. (See the documentation for the particular network you are using to determine the network addresses you need.)

There are at least two lines in the rfmaster file per domain name server: one p and one a line, to define the primary and its network address. There should also be at least one secondary name server in each domain.

This file is created and maintained on the primary domain name server. When a machine other than the primary tries to start Remote File Sharing, this file is read to determine the address of the primary. If rfmaster is missing, the -p option of rfstart must be used to identify the primary. After that, a copy of the primary's rfmaster file is automatically placed on the machine.

Domains not served by the primary can also be listed in the rfmaster file. By adding primary, secondary, and address information for other domains on a network, machines served by the primary will be able to share resources with machines in other domains.

A primary name server may be a primary for more than one domain. However, the secondaries must then also be the same for each domain served by the primary.
Example

An example of an `rfmaster` file is shown below. (The network address examples, `comp1.serve` and `comp2.serve`, are STARLAN network addresses.)

```
ccs      p      ccs.comp1
ccs      s      ccs.comp2
ccs.comp2 a      comp2.serve
ccs.comp1 a      comp1.serve
```

NOTE: If a line in the `rfmaster` file begins with a `#` character, the entire line will be treated as a comment.

FILES

`/usr/nserv/rfcmaster`

SEE ALSO

`rfstart(1M)` in the *System Administrator's Reference Manual*. 
NAME
sccsfile — format of SCCS file

DESCRIPTION
An SCCS (Source Code Control System) file is an ASCII file. It consists of six logical parts: the checksum, the delta table (contains information about each delta), user names (contains login names and/or numerical group IDs of users who may add deltas), flags (contains definitions of internal keywords), comments (contains arbitrary descriptive information about the file), and the body (contains the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as the control character and will be represented graphically as @. Any line described below which is not depicted as beginning with the control character is prevented from beginning with the control character.

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

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

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

@hDDDDD

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

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

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

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

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

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

The @e line ends the delta table entry.

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

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

@f <flag> <optional text>

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

The t flag defines the replacement for the %Y% identification keyword. The v flag controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program. The i flag controls the warning/error aspect of the "No id keywords" message. When the i flag is not present, this message is only a warning; when the i flag is present, this message will cause a "fatal" error (the file will not be gotten, or the delta will not be made). When the b flag is present the -b keyletter may be used on the get command to cause a branch in the delta tree. The m flag defines the first choice
for the replacement text of the %M% identification keyword. The f flag defines the "floor" release; the release below which no deltas may be added. The c flag defines the "ceiling" release; the release above which no deltas may be added. The d flag defines the default SID to be used when none is specified on a get command. The n flag causes delta to insert a "null" delta (a delta that applies no changes) in those releases that are skipped when a delta is made in a new release (e.g., when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the n flag causes skipped releases to be completely empty. The j flag causes get to allow concurrent edits of the same base SID. The I flag defines a list of releases that are locked against editing [get(1) with the -e keyletter]. The q flag defines the replacement for the %Q% identification keyword. The z flag is used in certain specialized interface programs. Comments Arbitrary text is surrounded by the bracketing lines @t and @T. The comments section typically will contain a description of the file's purpose.

Body

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

@I DDDDD
@D DDDDD
@E DDDDD

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

SEE ALSO

admin(1), delta(1), get(1), prs(1).
NAME
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_Innopr;      /* 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 is somewhat different. It 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_Innopr, 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() will copy the contents of the screen into a file. The format of the screen image is as 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.

<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 will be 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

syms – common object file symbol table format

SYNOPSIS

#include <syms.h>

DESCRIPTION

Common object files contain information to support symbolic software testing [see sdb(1)]. 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 given below.

```c
#define SYMNMLEN 8
#define FLINMLEN 14
#define DIMNUM 4

struct syment {
    union {
        char _n_name[SYMNMLEN]; /* symbol name */
        struct {
            long _n_zeroes; /* == 0L when in string table */
            long _n_offset; /* location of name in table */
        } _n.n;
        char *n_nptr[2]; /* allows overlaying */
    } _n;
};
```

574
Meaningful values and explanations for them are given in both syms.h and Common Object File Format. Anyone who needs to interpret the entries should seek more information in these sources. 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 short x_lno;
                unsigned short x_size;
            } x_lnsz;
            long x_fsize;
        } x_misc;
        union {
            struct {
                long x_lnptr;
                long x_endndx;
            } x_fcn;
            struct {
                unsigned short x_dimen[DIMNUM];
                x_ary;
                x_fcnary;
            } x_sym;
            unsigned short x_tvndx;
            struct {
                char x_fname[FILNMLEN];
            } x_file;
        }
    }
};
Indexes of symbol table entries begin at zero.

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

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

system — system configuration information table

DESCRIPTION

This file is used by the boot program to obtain configuration information that
cannot be obtained from the equipped device table (EDT) at system boot time.
This file generally contains a list of software drivers to include in the load,
the assignment of system devices such as pipedev and swapdev, as well as instruc-
tions for manually overriding the drivers selected by the self-configuring boot
process.

The syntax of the system file is given below. The parser for the /etc/system
file is case sensitive. All upper case strings in the syntax below should be upper
case in the /etc/system file as well. Nonterminal symbols are enclosed in angle
brackets "< >" while optional arguments are enclosed in square brackets "[]".
Ellipses "..." indicate optional repetition of the argument for that line.

\[\text{<fname>} ::= \text{pathname} \]
\[\text{<string>} ::= \text{driver file name from /boot or EDT entry name} \]
\[\text{<device>} ::= \text{special device name | DEV(<major>, <minor>)} \]
\[\text{<major>} ::= \text{<number>} \]
\[\text{<minor>} ::= \text{<number>} \]
\[\text{<number>} ::= \text{decimal, octal or hex literal} \]

The lines listed below may appear in any order. Blank lines may be inserted at
any point. Comment lines must begin with an asterisk. Entries for EXCLUDE
and INCLUDE are cumulative. For all other entries, the last line to appear in
the file is used -- any earlier entries are ignored.

\[\text{BOOT: <fname>} \]
\hspace{1cm} specifies the kernel a.out file to be booted; if the file is
\hspace{1cm} fully resolved [such as that produced by the mkunix(1M)
\hspace{1cm} program] then all other lines in the system file have no
\hspace{1cm} effect.

\[\text{EXCLUDE: [ <string> ] ...} \]
\hspace{1cm} specifies drivers to exclude from the load even if the
device is found in the EDT.

\[\text{INCLUDE: [ <string>[<number>] ] ...} \]
\hspace{1cm} specifies software drivers or loadable modules to be
\hspace{1cm} included in the load. This is necessary to include the
drivers for software "devices". The optional <number> (parenthesis required) specifies the number of "devices" to
\hspace{1cm} be controlled by the driver (defaults to 1). This number
\hspace{1cm} corresponds to the builtin variable #c which may be
\hspace{1cm} referred to by expressions in part one of the /etc/master
\hspace{1cm} file.

\[\text{ROOTDEV: <device>} \]
\hspace{1cm} identifies the device containing the root file system.
SWAPDEV: <device> <number> <number>
identifies the device to be used as swap space, the block
number the swap space starts at, and the number of swap
blocks available.

PIPEDEV: <device>
identifies the device to be used for pipe space.

FILES
/etc/system

SEE ALSO
master(4),
crash(1M), mkunix(1M), mkboot(1M) in the System Administrator's Reference Manual.
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 UNIX system direc-
tory, 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 will be 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 cancelled 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 NUL
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 cancelled.
The capabilities are in the same order as the file <term.h>.

Between the boolean section and the number section, a null byte will be inserted, if necessary, to ensure that the number section begins on an even byte.
All short integers are aligned on a short word boundary.
The numbers section is similar to the 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\ttty37\AT&T model 37 teletype,
hc, os, xon,
  bel=`G, cr=`r, cub1=`b, cud1=`n, cuu1=`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 \AT&T model 1
0000040  3 7 t e l e t y p e \0 \0 \0 \0 \0 \0
0000060  \0 \0 \0 001 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000100  001 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000120  377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 & \0
0000140  \0 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000160  377 377 \t 0 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000200  377 377 0 \0 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
  \0
  * \0
  * 0001160  \tty37\AT&T model 37 7
0001200  t t y 3 7 \AT&T mode
0001220  t y 3 7 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
```
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

curses(3X), terminfo(4), term(5).
Chapter 10 of the Programmer's Guide.
NAME
terminfo — terminal capability data base

SYNOPSIS
/usr/lib/terminfo/?/•

DESCRIPTION
terminfo is a compiled database (see tic(1M)) describing the capabilities of terminals. Terminals are described in terminfo source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used by applications programs, such as vi(1) and curses(3X), so they can work with a variety of terminals without changes to the programs. To obtain the source description for a terminal, use the −I option of infocmp(1M).

Entries in terminfo source files consist of a number of comma-separated fields. White space after each comma is ignored. The first line of each terminal description in the terminfo database gives the name by which terminfo 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 TERM 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. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. See term(5) for examples and more information on choosing names and synonyms.

CAPABILITIES
In the table below, the Variable is the name by which the C programmer (at the terminfo level) accesses the capability. The Capname is the short name for this variable used in the text of the database. It is used by a person updating the database and by the tput(1) command when asking what the value of the capability is for a particular terminal. The Termcap Code is a two-letter code that corresponds to the old termcap capability name.

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification.

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 table below, have names beginning with key_. The following indicators may appear at the end of the Description for a variable.
TERMINFO(4) (Terminal Information Utilities) TERMINFO(4)

(G) indicates that the string is passed through `tparm()` with parameters (parms) as given (#i).

(*) indicates that padding may be based on the number of lines affected.

(#) indicates the i\text{th} parameter.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Cap-name</th>
<th>Termcap Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Booleans:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>auto_left_margin</td>
<td>bw</td>
<td>bw</td>
<td>cub1 wraps from column 0 to last column</td>
</tr>
<tr>
<td>auto_right_margin</td>
<td>am</td>
<td>am</td>
<td>Terminal has automatic margins</td>
</tr>
<tr>
<td>no_esc_ctlc</td>
<td>xsb</td>
<td>xb</td>
<td>Beehive (f1=escape, f2=ctrl C)</td>
</tr>
<tr>
<td>ceol_standout_glitch</td>
<td>xhp</td>
<td>xs</td>
<td>Standout not erased by overwriting (hp)</td>
</tr>
<tr>
<td>eat_newline_glitch</td>
<td>xenl</td>
<td>xn</td>
<td>Newline ignored after 80 cols (Concept)</td>
</tr>
<tr>
<td>erase_overstrike</td>
<td>eo</td>
<td>eo</td>
<td>Can erase overstrikes with a blank</td>
</tr>
<tr>
<td>generic_type</td>
<td>gn</td>
<td>gn</td>
<td>Generic line type (e.g. dialup, switch).</td>
</tr>
<tr>
<td>hard_copy</td>
<td>hc</td>
<td>HC</td>
<td>Hardcopy terminal</td>
</tr>
<tr>
<td>hard_cursor</td>
<td>chts</td>
<td></td>
<td>Cursor is hard to see</td>
</tr>
<tr>
<td>has_meta_key</td>
<td>km</td>
<td>km</td>
<td>Has a meta key (shift, sets parity bit)</td>
</tr>
<tr>
<td>has_status_line</td>
<td>hs</td>
<td>hs</td>
<td>Has extra &quot;status line&quot;</td>
</tr>
<tr>
<td>insert_null_glitch</td>
<td>in</td>
<td>in</td>
<td>Insert mode distinguishes nulls</td>
</tr>
<tr>
<td>memory_above</td>
<td>da</td>
<td>da</td>
<td>Display may be retained above the screen</td>
</tr>
<tr>
<td>memory_below</td>
<td>db</td>
<td>db</td>
<td>Display may be retained below the screen</td>
</tr>
<tr>
<td>move_insert_mode</td>
<td>mir</td>
<td>mi</td>
<td>Safe to move while in insert mode</td>
</tr>
<tr>
<td>move_standout_mode</td>
<td>msr</td>
<td>ms</td>
<td>Safe to move in standout modes</td>
</tr>
<tr>
<td>needs_xon_xoff</td>
<td>nxon</td>
<td>nx</td>
<td>Padding won't work, xon/xoff required</td>
</tr>
<tr>
<td>non_rev_rmcup</td>
<td>nrmc</td>
<td>NR</td>
<td>smcup does not reverse rmcup</td>
</tr>
<tr>
<td>no_pad_char</td>
<td>npc</td>
<td>NP</td>
<td>Pad character doesn't exist</td>
</tr>
<tr>
<td>over_strike</td>
<td>os</td>
<td>os</td>
<td>Terminal overstrikes on hard-copy terminal</td>
</tr>
<tr>
<td>prtr_silent</td>
<td>mc5i</td>
<td>5i</td>
<td>Printer won't echo on screen.</td>
</tr>
<tr>
<td>status_line_esc_ok</td>
<td>eslok</td>
<td>es</td>
<td>Escape can be used on the status line</td>
</tr>
<tr>
<td>dest_tabs_magic_smso</td>
<td>xt</td>
<td>xt</td>
<td>Destructive tabs, magic smso char (t1061)</td>
</tr>
<tr>
<td>tilde_glitch</td>
<td>hz</td>
<td>hz</td>
<td>Hazeltine; can't print tildes(\text{t})</td>
</tr>
<tr>
<td>transparent_underline</td>
<td>ul</td>
<td>ul</td>
<td>Underline character overstrikes</td>
</tr>
<tr>
<td>xon_xoff</td>
<td>xon</td>
<td>xo</td>
<td>Terminal uses xon/xoff handshaking</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Numbers:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>columns</td>
<td>cols</td>
<td>co</td>
<td>Number of columns in a line</td>
</tr>
<tr>
<td>init_tabs</td>
<td>it</td>
<td>it</td>
<td>Tabs initially every # spaces.</td>
</tr>
<tr>
<td>label_height</td>
<td>lh</td>
<td>lh</td>
<td>Number of rows in each label</td>
</tr>
<tr>
<td>label_width</td>
<td>lw</td>
<td>lw</td>
<td>Number of cols in each label</td>
</tr>
<tr>
<td>lines</td>
<td>lines</td>
<td>li</td>
<td>Number of lines on screen or page</td>
</tr>
<tr>
<td>lines_of_memory</td>
<td>lm</td>
<td>lm</td>
<td>Lines of memory if &gt; lines; 0 means varies</td>
</tr>
<tr>
<td>magic_cookie_glitch</td>
<td>xmc</td>
<td>sg</td>
<td>Number blank chars left by smso or rmcup</td>
</tr>
<tr>
<td>num_labels</td>
<td>nlab</td>
<td>NL</td>
<td>Number of labels on screen (start at 1)</td>
</tr>
<tr>
<td>padding_baud_rate</td>
<td>pb</td>
<td>pb</td>
<td>Lowest baud rate where padding needed</td>
</tr>
<tr>
<td>virtual_terminal</td>
<td>vt</td>
<td>vt</td>
<td>Virtual terminal number (UNIX system)</td>
</tr>
<tr>
<td>width_status_line</td>
<td>wsl</td>
<td>ws</td>
<td>Number of columns in status line</td>
</tr>
</tbody>
</table>

583
### Strings:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>acs_chars</code></td>
<td>Graphic charset pairs aAbBcC - def=vt100+</td>
</tr>
<tr>
<td><code>back_tab</code></td>
<td>Back tab</td>
</tr>
<tr>
<td><code>bell</code></td>
<td>Audible signal (bell)</td>
</tr>
<tr>
<td><code>carriage_return</code></td>
<td>Carriage return (*)</td>
</tr>
<tr>
<td><code>change_scroll_region</code></td>
<td>Change to lines #1 thru #2 (vt100) (G)</td>
</tr>
<tr>
<td><code>char_padding</code></td>
<td>Like <code>ip</code> but when in replace mode</td>
</tr>
<tr>
<td><code>clear_all_tabs</code></td>
<td>Clear all tab stops</td>
</tr>
<tr>
<td><code>clear_left</code></td>
<td>Clear left and right soft margins</td>
</tr>
<tr>
<td><code>clear_screen</code></td>
<td>Clear screen and home cursor (*)</td>
</tr>
<tr>
<td><code>clear_to_end</code></td>
<td>Clear to end of line</td>
</tr>
<tr>
<td><code>clear_to_beginning</code></td>
<td>Clear to beginning of line, inclusive</td>
</tr>
<tr>
<td><code>column_address</code></td>
<td>Horizontal position absolute (G)</td>
</tr>
<tr>
<td><code>command_character</code></td>
<td>Term. settable cmd char in prototype</td>
</tr>
<tr>
<td><code>cursor_address</code></td>
<td>Cursor motion to row #1 col #2 (G)</td>
</tr>
<tr>
<td><code>cursor_down</code></td>
<td>Down one line</td>
</tr>
<tr>
<td><code>cursor_home</code></td>
<td>Home cursor (if no <code>cup</code>)</td>
</tr>
<tr>
<td><code>cursor_invisible</code></td>
<td>Make cursor invisible</td>
</tr>
<tr>
<td><code>cursor_left</code></td>
<td>Move cursor left one space</td>
</tr>
<tr>
<td><code>cursor_mem_address</code></td>
<td>Memory relative cursor addressing (G)</td>
</tr>
<tr>
<td><code>cursor_normal</code></td>
<td>Make cursor appear normal (undo <code>vs/vi</code>)</td>
</tr>
<tr>
<td><code>cursor_right</code></td>
<td>Non-destructive space (cursor right)</td>
</tr>
<tr>
<td><code>cursor_to_line</code></td>
<td>Last line, first column (if no <code>cup</code>)</td>
</tr>
<tr>
<td><code>cursor_up</code></td>
<td>Upline (cursor up)</td>
</tr>
<tr>
<td><code>cursor_visible</code></td>
<td>Make cursor very visible</td>
</tr>
<tr>
<td><code>delete_character</code></td>
<td>Delete character (*)</td>
</tr>
<tr>
<td><code>delete_line</code></td>
<td>Delete line (*)</td>
</tr>
<tr>
<td><code>dis_status_line</code></td>
<td>Disable status line</td>
</tr>
<tr>
<td><code>down_half_line</code></td>
<td>Half-line down (forward 1/2 linefeed)</td>
</tr>
<tr>
<td><code>enabacs</code></td>
<td>Enable alternate char set</td>
</tr>
<tr>
<td><code>enter_alt_charset_mode</code></td>
<td>Start alternate character set; Turn on automatic margins</td>
</tr>
<tr>
<td><code>enter_am_mode</code></td>
<td>Turn on automatic margins</td>
</tr>
<tr>
<td><code>enter_blink_mode</code></td>
<td>Turn on blinking</td>
</tr>
<tr>
<td><code>enter_bold_mode</code></td>
<td>Turn on bold (extra bright) mode</td>
</tr>
<tr>
<td><code>enter_ca_mode</code></td>
<td>String to begin programs that use <code>cup</code></td>
</tr>
<tr>
<td><code>enter_delete_mode</code></td>
<td>Delete mode (enter)</td>
</tr>
<tr>
<td><code>enter_dim_mode</code></td>
<td>Turn on half-bright mode</td>
</tr>
<tr>
<td><code>enter_insert_mode</code></td>
<td>Insert mode (enter);</td>
</tr>
<tr>
<td><code>enter_protected_mode</code></td>
<td>Turn on protected mode</td>
</tr>
<tr>
<td><code>enter_reverse_mode</code></td>
<td>Turn on reverse video mode</td>
</tr>
<tr>
<td><code>enter_secure_mode</code></td>
<td>Turn on blank mode (chars invisible)</td>
</tr>
<tr>
<td><code>enter_standout_mode</code></td>
<td>Begin standout mode</td>
</tr>
<tr>
<td><code>enter_underline_mode</code></td>
<td>Start underscore mode</td>
</tr>
<tr>
<td><code>exit_alt_charset_mode</code></td>
<td>Turn on xon/xoff handshaking</td>
</tr>
<tr>
<td><code>exit_am_mode</code></td>
<td>End alternate character set</td>
</tr>
<tr>
<td><code>exit_attribute_mode</code></td>
<td>Turn off automatic margins</td>
</tr>
<tr>
<td><code>exit_standout_mode</code></td>
<td>Turn off all attributes</td>
</tr>
</tbody>
</table>
TERMINFO(4) (Terminal Information Utilities) TERMINFO(4)

**exit_ca_mode**
rsync
ted
String to end programs that use **cup**

**exit_delete_mode**
rmr
de
End delete mode

**exit_insert_mode**
rmi
ri
End insert mode;

**exit_standout_mode**
rmso
se
End standout mode

**exit_underline_mode**
rmul
ue
End underscore mode

**exit_xon_mode**
rmxon
RX
Turn off xon/xoff handshaking

**flash_screen**
flash
vb
Visible bell (may not move cursor)

**form_feed**
ff
ff
Hard copy terminal page eject (*)

**from_status_line**
fs1
fs
Return from status line

**init_1string**
is1
i1
Terminal initialization string

**init_2string**
is2
is
Terminal initialization string

**init_3string**
is3
i3
Terminal initialization string

**init_file**
if
if
Name of initialization file containing is

**init_prog**
ipro
p
Path name of program for init.

**insert_character**
ich1
ic
Insert character

**insert_line**
il1
il
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**
kb
kb
KEY_BACKSPACE, 0407, Sent by backspace key

**key_beg**
kbeg
@1
KEY_BEG, 0542, Sent by beg(inning) key

**key_btab**
kbet
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**
kcanc
@2
KEYCANCEL, 0543, Sent by cancel key

**key_catab**
ktbc
ka
KEYCATAB, 0526, Sent by clear-all-tabs key

**key_clear**
kcrl
kC
KEYCLEAR, 0515, Sent by clear-screen or erase key

**key_close**
kclo
@3
KEYCLOSE, 0544, Sent by close key

**key_command**
kmnd
@4
KEYCOMMAND, 0545, Sent by cmd (command) key

**key_copy**
kcpi
@5
KEYCOPY, 0546, Sent by copy key

**key_create**
kcrt
@6
KEYCREATE, 0547, Sent by create key

**key_ctab**
kctab
kt
KEYCTAB, 0525, Sent by clear-tab key

**key_dc**
kdc
KD
KEY_DC, 0512, Sent by delete-character key

**key_dl**
kdl1
kL
KEY_DL, 0510, Sent by delete-line key

**key_down**
kcd1
kd
KEY_DOWN, 0402, Sent by terminal down-arrow key

**key_eic**
krmir
kM
KEYEIC, 0514, Sent by **rmir** or **smir** in insert mode

**key_end**
kend
@7
KEYEND, 0550, Sent by end key

**key_enter**
kent
@8
KEYENTER, 0527, Sent by enter/send key

**key_eol**
kel
kE
KEY_EOL, 0517, Sent by clear-to-end-of-line key

**key_eos**
ked
kS
KEYEOS, 0516, Sent by clear-to-end-of-screen key

**key_exit**
kexit
@9
KEYEXIT, 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

**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
<table>
<thead>
<tr>
<th>key</th>
<th>function_key</th>
<th>sent_by</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_f7</td>
<td>kf7</td>
<td>k7</td>
<td>KEY_F(7), 0417, Sent by function key f7</td>
</tr>
<tr>
<td>key_f8</td>
<td>kf8</td>
<td>k8</td>
<td>KEY_F(8), 0420, Sent by function key f8</td>
</tr>
<tr>
<td>key_f9</td>
<td>kf9</td>
<td>k9</td>
<td>KEY_F(9), 0421, Sent by function key f9</td>
</tr>
<tr>
<td>key_f10</td>
<td>kf10</td>
<td>k;</td>
<td>KEY_F(10), 0422, Sent by function key f10</td>
</tr>
<tr>
<td>key_f11</td>
<td>kf11</td>
<td>F1</td>
<td>KEY_F(11), 0423, Sent by function key f11</td>
</tr>
<tr>
<td>key_f12</td>
<td>kf12</td>
<td>F2</td>
<td>KEY_F(12), 0424, Sent by function key f12</td>
</tr>
<tr>
<td>key_f13</td>
<td>kf13</td>
<td>F3</td>
<td>KEY_F(13), 0425, Sent by function key f13</td>
</tr>
<tr>
<td>key_f14</td>
<td>kf14</td>
<td>F4</td>
<td>KEY_F(14), 0426, Sent by function key f14</td>
</tr>
<tr>
<td>key_f15</td>
<td>kf15</td>
<td>F5</td>
<td>KEY_F(15), 0427, Sent by function key f15</td>
</tr>
<tr>
<td>key_f16</td>
<td>kf16</td>
<td>F6</td>
<td>KEY_F(16), 0430, Sent by function key f16</td>
</tr>
<tr>
<td>key_f17</td>
<td>kf17</td>
<td>F7</td>
<td>KEY_F(17), 0431, Sent by function key f17</td>
</tr>
<tr>
<td>key_f18</td>
<td>kf18</td>
<td>F8</td>
<td>KEY_F(18), 0432, Sent by function key f18</td>
</tr>
<tr>
<td>key_f19</td>
<td>kf19</td>
<td>F9</td>
<td>KEY_F(19), 0433, Sent by function key f19</td>
</tr>
<tr>
<td>key_f20</td>
<td>kf20</td>
<td>FA</td>
<td>KEY_F(20), 0434, Sent by function key f20</td>
</tr>
<tr>
<td>key_f21</td>
<td>kf21</td>
<td>FB</td>
<td>KEY_F(21), 0435, Sent by function key f21</td>
</tr>
<tr>
<td>key_f22</td>
<td>kf22</td>
<td>FC</td>
<td>KEY_F(22), 0436, Sent by function key f22</td>
</tr>
<tr>
<td>key_f23</td>
<td>kf23</td>
<td>FD</td>
<td>KEY_F(23), 0437, Sent by function key f23</td>
</tr>
<tr>
<td>key_f24</td>
<td>kf24</td>
<td>FE</td>
<td>KEY_F(24), 0440, Sent by function key f24</td>
</tr>
<tr>
<td>key_f25</td>
<td>kf25</td>
<td>FF</td>
<td>KEY_F(25), 0441, Sent by function key f25</td>
</tr>
<tr>
<td>key_f26</td>
<td>kf26</td>
<td>FG</td>
<td>KEY_F(26), 0442, Sent by function key f26</td>
</tr>
<tr>
<td>key_f27</td>
<td>kf27</td>
<td>FH</td>
<td>KEY_F(27), 0443, Sent by function key f27</td>
</tr>
<tr>
<td>key_f28</td>
<td>kf28</td>
<td>FI</td>
<td>KEY_F(28), 0444, Sent by function key f28</td>
</tr>
<tr>
<td>key_f29</td>
<td>kf29</td>
<td>FJ</td>
<td>KEY_F(29), 0445, Sent by function key f29</td>
</tr>
<tr>
<td>key_f30</td>
<td>kf30</td>
<td>FK</td>
<td>KEY_F(30), 0446, Sent by function key f30</td>
</tr>
<tr>
<td>key_f31</td>
<td>kf31</td>
<td>FL</td>
<td>KEY_F(31), 0447, Sent by function key f31</td>
</tr>
<tr>
<td>key_f32</td>
<td>kf32</td>
<td>FM</td>
<td>KEY_F(32), 0450, Sent by function key f32</td>
</tr>
<tr>
<td>key_f33</td>
<td>kf33</td>
<td>FN</td>
<td>KEY_F(33), 0451, Sent by function key f33</td>
</tr>
<tr>
<td>key_f34</td>
<td>kf34</td>
<td>FO</td>
<td>KEY_F(34), 0452, Sent by function key f34</td>
</tr>
<tr>
<td>key_f35</td>
<td>kf35</td>
<td>FP</td>
<td>KEY_F(35), 0453, Sent by function key f35</td>
</tr>
<tr>
<td>key_f36</td>
<td>kf36</td>
<td>FQ</td>
<td>KEY_F(36), 0454, Sent by function key f36</td>
</tr>
<tr>
<td>key_f37</td>
<td>kf37</td>
<td>FR</td>
<td>KEY_F(37), 0455, Sent by function key f37</td>
</tr>
<tr>
<td>key_f38</td>
<td>kf38</td>
<td>FS</td>
<td>KEY_F(38), 0456, Sent by function key f38</td>
</tr>
<tr>
<td>key_f39</td>
<td>kf39</td>
<td>FT</td>
<td>KEY_F(39), 0457, Sent by function key f39</td>
</tr>
<tr>
<td>key_f40</td>
<td>kf40</td>
<td>FU</td>
<td>KEY_F(40), 0460, Sent by function key f40</td>
</tr>
<tr>
<td>key_f41</td>
<td>kf41</td>
<td>FV</td>
<td>KEY_F(41), 0461, Sent by function key f41</td>
</tr>
<tr>
<td>key_f42</td>
<td>kf42</td>
<td>FW</td>
<td>KEY_F(42), 0462, Sent by function key f42</td>
</tr>
<tr>
<td>key_f43</td>
<td>kf43</td>
<td>FX</td>
<td>KEY_F(43), 0463, Sent by function key f43</td>
</tr>
<tr>
<td>key_f44</td>
<td>kf44</td>
<td>FY</td>
<td>KEY_F(44), 0464, Sent by function key f44</td>
</tr>
<tr>
<td>key_f45</td>
<td>kf45</td>
<td>FZ</td>
<td>KEY_F(45), 0465, Sent by function key f45</td>
</tr>
<tr>
<td>key_f46</td>
<td>kf46</td>
<td>Fa</td>
<td>KEY_F(46), 0466, Sent by function key f46</td>
</tr>
<tr>
<td>key_f47</td>
<td>kf47</td>
<td>Fb</td>
<td>KEY_F(47), 0467, Sent by function key f47</td>
</tr>
<tr>
<td>key_f48</td>
<td>kf48</td>
<td>Fc</td>
<td>KEY_F(48), 0470, Sent by function key f48</td>
</tr>
<tr>
<td>key_f49</td>
<td>kf49</td>
<td>Fd</td>
<td>KEY_F(49), 0471, Sent by function key f49</td>
</tr>
<tr>
<td>key_f50</td>
<td>kf50</td>
<td>Fe</td>
<td>KEY_F(50), 0472, Sent by function key f50</td>
</tr>
<tr>
<td>key_f51</td>
<td>kf51</td>
<td>Ff</td>
<td>KEY_F(51), 0473, Sent by function key f51</td>
</tr>
<tr>
<td>key_f52</td>
<td>kf52</td>
<td>Fg</td>
<td>KEY_F(52), 0474, Sent by function key f52</td>
</tr>
<tr>
<td>key_f53</td>
<td>kf53</td>
<td>Fh</td>
<td>KEY_F(53), 0475, Sent by function key f53</td>
</tr>
<tr>
<td>key_f54</td>
<td>kf54</td>
<td>Fi</td>
<td>KEY_F(54), 0476, Sent by function key f54</td>
</tr>
<tr>
<td>key_f55</td>
<td>kf55</td>
<td>Fj</td>
<td>KEY_F(55), 0477, Sent by function key f55</td>
</tr>
<tr>
<td>Term</td>
<td>Code</td>
<td>Description</td>
<td>Code</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>---------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>key_f56</td>
<td>kf56</td>
<td>Sent by function key f56</td>
<td>Fk</td>
</tr>
<tr>
<td>key_f57</td>
<td>kf57</td>
<td>Sent by function key f57</td>
<td>Fl</td>
</tr>
<tr>
<td>key_f58</td>
<td>kf58</td>
<td>Sent by function key f58</td>
<td>Fm</td>
</tr>
<tr>
<td>key_f59</td>
<td>kf59</td>
<td>Sent by function key f59</td>
<td>Fn</td>
</tr>
<tr>
<td>key_f60</td>
<td>kf60</td>
<td>Sent by function key f60</td>
<td>Fo</td>
</tr>
<tr>
<td>key_f61</td>
<td>kf61</td>
<td>Sent by function key f61</td>
<td>Fp</td>
</tr>
<tr>
<td>key_f62</td>
<td>kf62</td>
<td>Sent by function key f62</td>
<td>Fq</td>
</tr>
<tr>
<td>key_f63</td>
<td>kf63</td>
<td>Sent by function key f63</td>
<td>Fr</td>
</tr>
<tr>
<td>key_find</td>
<td>kfnd</td>
<td>Sent by find key</td>
<td>@O</td>
</tr>
<tr>
<td>key_help</td>
<td>khelp</td>
<td>Sent by help key</td>
<td>%1</td>
</tr>
<tr>
<td>key_home</td>
<td>khome</td>
<td>Sent by home key</td>
<td>kh</td>
</tr>
<tr>
<td>key_ic</td>
<td>kich1</td>
<td>Sent by ins-char/enter ins-mode key</td>
<td>kI</td>
</tr>
<tr>
<td>key_il</td>
<td>kil1</td>
<td>Sent by insert-line key</td>
<td>kA</td>
</tr>
<tr>
<td>key_left</td>
<td>kcub1</td>
<td>Sent by terminal left-arrow key</td>
<td>kl</td>
</tr>
<tr>
<td>key_ll</td>
<td>kl</td>
<td>Sent by home-down key</td>
<td>kH</td>
</tr>
<tr>
<td>key_mark</td>
<td>kmrk</td>
<td>Sent by mark key</td>
<td>%2</td>
</tr>
<tr>
<td>key_message</td>
<td>kmsg</td>
<td>Sent by message key</td>
<td>%3</td>
</tr>
<tr>
<td>key_move</td>
<td>kmov</td>
<td>Sent by move key</td>
<td>%4</td>
</tr>
<tr>
<td>key_next</td>
<td>knxt</td>
<td>Sent by next-object key</td>
<td>%5</td>
</tr>
<tr>
<td>key_npage</td>
<td>knp</td>
<td>Sent by next-page key</td>
<td>kN</td>
</tr>
<tr>
<td>key_open</td>
<td>kopn</td>
<td>Sent by open key</td>
<td>%6</td>
</tr>
<tr>
<td>key_options</td>
<td>kopt</td>
<td>Sent by options key</td>
<td>%7</td>
</tr>
<tr>
<td>key_ppage</td>
<td>kpp</td>
<td>Sent by previous-page key</td>
<td>kP</td>
</tr>
<tr>
<td>key_previous</td>
<td>kprv</td>
<td>Sent by previous-object key</td>
<td>%8</td>
</tr>
<tr>
<td>key_print</td>
<td>kprt</td>
<td>Sent by print or copy key</td>
<td>%9</td>
</tr>
<tr>
<td>key_redo</td>
<td>kredo</td>
<td>Sent by redo key</td>
<td>%0</td>
</tr>
<tr>
<td>key_reference</td>
<td>kref</td>
<td>Sent by (reference) key</td>
<td>&amp;1</td>
</tr>
<tr>
<td>key_refresh</td>
<td>krefr</td>
<td>Sent by refresh key</td>
<td>&amp;2</td>
</tr>
<tr>
<td>key_replace</td>
<td>krpl</td>
<td>Sent by replace key</td>
<td>&amp;3</td>
</tr>
<tr>
<td>key_restart</td>
<td>krst</td>
<td>Sent by restart key</td>
<td>&amp;4</td>
</tr>
<tr>
<td>key_resume</td>
<td>kres</td>
<td>Sent by resume key</td>
<td>&amp;5</td>
</tr>
<tr>
<td>key_right</td>
<td>kcf1</td>
<td>Sent by terminal right-arrow key</td>
<td>kr</td>
</tr>
<tr>
<td>key_save</td>
<td>ksave</td>
<td>Sent by save key</td>
<td>&amp;6</td>
</tr>
<tr>
<td>key_sbeg</td>
<td>kbeg</td>
<td>Sent by shifted beginning key</td>
<td>&amp;9</td>
</tr>
<tr>
<td>key_scancel</td>
<td>kscanl</td>
<td>Sent by shifted cancel key</td>
<td>&amp;0</td>
</tr>
<tr>
<td>key_scommand</td>
<td>kcmd</td>
<td>Sent by shifted command key</td>
<td>*1</td>
</tr>
<tr>
<td>key_scopy</td>
<td>kcpy</td>
<td>Sent by shifted copy key</td>
<td>*2</td>
</tr>
<tr>
<td>key_screate</td>
<td>kcret</td>
<td>Sent by shifted create key</td>
<td>*3</td>
</tr>
<tr>
<td>key_sdc</td>
<td>kdc</td>
<td>Sent by shifted delete-char key</td>
<td>*4</td>
</tr>
<tr>
<td>key_sdl</td>
<td>kdl</td>
<td>Sent by shifted delete-line key</td>
<td>*5</td>
</tr>
<tr>
<td>key_select</td>
<td>kslt</td>
<td>Sent by select key</td>
<td>*6</td>
</tr>
<tr>
<td>key_send</td>
<td>ksend</td>
<td>Sent by shifted end key</td>
<td>*7</td>
</tr>
<tr>
<td>key_seol</td>
<td>kseol</td>
<td>Sent by shifted clear-line key</td>
<td>*8</td>
</tr>
<tr>
<td>key_sexit</td>
<td>kexit</td>
<td>Sent by shifted exit key</td>
<td>*9</td>
</tr>
<tr>
<td>key_sf</td>
<td>kind</td>
<td>Sent by scroll-forward/down key</td>
<td>kF</td>
</tr>
<tr>
<td>key_sfind</td>
<td>kfind</td>
<td>Sent by shifted find key</td>
<td>*0</td>
</tr>
<tr>
<td>key_shelp</td>
<td>khelp</td>
<td>Sent by shifted help key</td>
<td>#1</td>
</tr>
<tr>
<td>key_shome</td>
<td>khome</td>
<td>Sent by shifted home key</td>
<td>#2</td>
</tr>
<tr>
<td>key_sic</td>
<td>kic</td>
<td>Sent by shifted input key</td>
<td>#3</td>
</tr>
</tbody>
</table>

587
<table>
<thead>
<tr>
<th>Command</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_sleft</td>
<td>kLFT</td>
<td>#4 KEY_SLEFT, 0611, Sent by shifted left-arrow key</td>
</tr>
<tr>
<td>key_smessage</td>
<td>kMSG</td>
<td>%a KEY_SMESAGE, 0612, Sent by shifted message key</td>
</tr>
<tr>
<td>key_smove</td>
<td>kMOV</td>
<td>%b KEY_SMOVE, 0613, Sent by shifted move key</td>
</tr>
<tr>
<td>key_snex</td>
<td>kNXT</td>
<td>%c KEY_SNEXT, 0614, Sent by shifted next key</td>
</tr>
<tr>
<td>key_soptions</td>
<td>kOPT</td>
<td>%d KEY_SOPTIONS, 0615, Sent by shifted options key</td>
</tr>
<tr>
<td>key_spervious</td>
<td>kPRV</td>
<td>%e KEY_SPREVIOUS, 0616, Sent by shifted prev key</td>
</tr>
<tr>
<td>key_sprint</td>
<td>kPRT</td>
<td>%f KEY_SPRINT, 0617, Sent by shifted print key</td>
</tr>
<tr>
<td>key_sr</td>
<td>kri</td>
<td>kR KEY_SR, 0521, Sent by scroll-backward/up key</td>
</tr>
<tr>
<td>key_sredo</td>
<td>kRDO</td>
<td>%g KEY_SREDO, 0620, Sent by shifted redo key</td>
</tr>
<tr>
<td>key_sreplace</td>
<td>kRPL</td>
<td>%h KEY_SREPLACE, 0621, Sent by shifted replace key</td>
</tr>
<tr>
<td>key_sright</td>
<td>kRIT</td>
<td>%i KEY_SRIGHT, 0622, Sent by shifted right-arrow key</td>
</tr>
<tr>
<td>key_srsume</td>
<td>kRES</td>
<td>%j KEY_SRsume, 0623, Sent by shifted resume key</td>
</tr>
<tr>
<td>key_ssave</td>
<td>kSAV</td>
<td>!1 KEY_SSAVE, 0624, Sent by shifted save key</td>
</tr>
<tr>
<td>key_ssuspend</td>
<td>kSPD</td>
<td>!2 KEY_SsUSPEND, 0625, Sent by shifted suspend key</td>
</tr>
<tr>
<td>key_stab</td>
<td>khts</td>
<td>kT KEY_STAB, 0524, Sent by set-tab key</td>
</tr>
<tr>
<td>key_sundo</td>
<td>kUND</td>
<td>!3 KEY_SUNDO, 0626, Sent by shifted undo key</td>
</tr>
<tr>
<td>key_suspend</td>
<td>kspd</td>
<td>&amp;7 KEY_SUSPEND, 0627, Sent by suspend key</td>
</tr>
<tr>
<td>key_undo</td>
<td>kund</td>
<td>&amp;8 KEY_UNDO, 0630, Sent by undo key</td>
</tr>
<tr>
<td>key_up</td>
<td>kcuu1</td>
<td>kU KEY_UP, 0403, Sent by terminal up-arrow key</td>
</tr>
<tr>
<td>keypad_local</td>
<td>rmkx</td>
<td>ke Out of “keypad-transmit” mode</td>
</tr>
<tr>
<td>keypad_xmit</td>
<td>smkx</td>
<td>ks Put terminal in “keypad-transmit” mode</td>
</tr>
<tr>
<td>lab_f0</td>
<td>lfo</td>
<td>10 Labels on function key f0 if not f0</td>
</tr>
<tr>
<td>lab_f1</td>
<td>lfo1</td>
<td>11 Labels on function key f1 if not f1</td>
</tr>
<tr>
<td>lab_f2</td>
<td>lfo2</td>
<td>12 Labels on function key f2 if not f2</td>
</tr>
<tr>
<td>lab_f3</td>
<td>lfo3</td>
<td>13 Labels on function key f3 if not f3</td>
</tr>
<tr>
<td>lab_f4</td>
<td>lfo4</td>
<td>14 Labels on function key f4 if not f4</td>
</tr>
<tr>
<td>lab_f5</td>
<td>lfo5</td>
<td>15 Labels on function key f5 if not f5</td>
</tr>
<tr>
<td>lab_f6</td>
<td>lfo6</td>
<td>16 Labels on function key f6 if not f6</td>
</tr>
<tr>
<td>lab_f7</td>
<td>lfo7</td>
<td>17 Labels on function key f7 if not f7</td>
</tr>
<tr>
<td>lab_f8</td>
<td>lfo8</td>
<td>18 Labels on function key f8 if not f8</td>
</tr>
<tr>
<td>lab_f9</td>
<td>lfo9</td>
<td>19 Labels on function key f9 if not f9</td>
</tr>
<tr>
<td>lab_f10</td>
<td>lfo10</td>
<td>la Labels on function key f10 if not f10</td>
</tr>
<tr>
<td>label_off</td>
<td>rmln</td>
<td>LF Turn off soft labels</td>
</tr>
<tr>
<td>label_on</td>
<td>smln</td>
<td>LO Turn on soft labels</td>
</tr>
<tr>
<td>meta_off</td>
<td>rmm</td>
<td>m0 Turn off “meta mode”</td>
</tr>
<tr>
<td>meta_on</td>
<td>smm</td>
<td>mm Turn on “meta mode” (8th bit)</td>
</tr>
<tr>
<td>newline</td>
<td>nel</td>
<td>nw Newline (behaves like cr followed by If)</td>
</tr>
<tr>
<td>pad_char</td>
<td>pad</td>
<td>pc Pad character (rather than null)</td>
</tr>
<tr>
<td>parm_dch</td>
<td>dch</td>
<td>DC Delete #1 chars (G*)</td>
</tr>
<tr>
<td>parm_delete_line</td>
<td>dl</td>
<td>DL Delete #1 lines (G*)</td>
</tr>
<tr>
<td>parm_down_cursor</td>
<td>cud</td>
<td>DO Move cursor down #1 lines. (G*)</td>
</tr>
<tr>
<td>parm_ich</td>
<td>ich</td>
<td>IC Insert #1 blank chars (G*)</td>
</tr>
<tr>
<td>parm_index</td>
<td>indn</td>
<td>SF Scroll forward #1 lines. (G)</td>
</tr>
<tr>
<td>parm_insert_line</td>
<td>il</td>
<td>AL Add #1 new blank lines (G*)</td>
</tr>
<tr>
<td>parm_left_cursor</td>
<td>cub</td>
<td>LE Move cursor left #1 spaces (G)</td>
</tr>
<tr>
<td>parm_right_cursor</td>
<td>cuf</td>
<td>RI Move cursor right #1 spaces. (G*)</td>
</tr>
<tr>
<td>parm_rindex</td>
<td>rin</td>
<td>SR Scroll backward #1 lines. (G)</td>
</tr>
<tr>
<td>parm_up_cursor</td>
<td>cuu</td>
<td>UP Move cursor up #1 lines. (G*)</td>
</tr>
<tr>
<td>pkey_key</td>
<td>pfkey</td>
<td>pk Prog funct key #1 to type string #2</td>
</tr>
</tbody>
</table>

588
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pkey_local</code></td>
<td>Prog funct key #1 to execute string #2</td>
</tr>
<tr>
<td><code>pkey_xmit</code></td>
<td>Prog funct key #1 to xmit string #2</td>
</tr>
<tr>
<td><code>plab_norm</code></td>
<td>Prog label #1 to show string #2</td>
</tr>
<tr>
<td><code>print_screen</code></td>
<td>Print contents of the screen</td>
</tr>
<tr>
<td><code>prtr_non</code></td>
<td>Turn on the printer for #1 bytes</td>
</tr>
<tr>
<td><code>prtr_off</code></td>
<td>Turn off the printer</td>
</tr>
<tr>
<td><code>prtr_on</code></td>
<td>Turn on the printer</td>
</tr>
<tr>
<td><code>repeat_char</code></td>
<td>Repeat char #1 #2 times (G*)</td>
</tr>
<tr>
<td><code>req_for_input</code></td>
<td>Send next input char (for ptys)</td>
</tr>
<tr>
<td><code>reset_1string</code></td>
<td>Reset terminal completely to sane modes</td>
</tr>
<tr>
<td><code>reset_2string</code></td>
<td>Reset terminal completely to sane modes</td>
</tr>
<tr>
<td><code>reset_3string</code></td>
<td>Reset terminal completely to sane modes</td>
</tr>
<tr>
<td><code>reset_file</code></td>
<td>Name of file containing reset string</td>
</tr>
<tr>
<td><code>restore_cursor</code></td>
<td>Restore cursor to position of last sc</td>
</tr>
<tr>
<td><code>row_address</code></td>
<td>Vertical position absolute (G)</td>
</tr>
<tr>
<td><code>save_cursor</code></td>
<td>Save cursor position</td>
</tr>
<tr>
<td><code>scroll_forward</code></td>
<td>Scroll text up</td>
</tr>
<tr>
<td><code>scroll_reverse</code></td>
<td>Scroll text down</td>
</tr>
<tr>
<td><code>set_attributes</code></td>
<td>Define the video attributes #1-#9 (G)</td>
</tr>
<tr>
<td><code>set_left_margin</code></td>
<td>Set soft left margin</td>
</tr>
<tr>
<td><code>set_right_margin</code></td>
<td>Set soft right margin</td>
</tr>
<tr>
<td><code>set_tab</code></td>
<td>Set a tab in all rows, current column.</td>
</tr>
<tr>
<td><code>set_window</code></td>
<td>Current window is lines #1-#2 cols #3-#4 (G)</td>
</tr>
<tr>
<td><code>tab</code></td>
<td>Tab to next 8 space hardware tab stop.</td>
</tr>
<tr>
<td><code>to_status_line</code></td>
<td>Go to status line, col #1 (G)</td>
</tr>
<tr>
<td><code>underline_char</code></td>
<td>Underscore one char and move past it</td>
</tr>
<tr>
<td><code>up_half_line</code></td>
<td>Half-line up (reverse 1/2 linefeed)</td>
</tr>
<tr>
<td><code>xoff_character</code></td>
<td>X-off character</td>
</tr>
<tr>
<td><code>xon_character</code></td>
<td>X-on character</td>
</tr>
</tbody>
</table>

**SAMPLE ENTRY**

The following entry, which describes the Concept-100 terminal, is among the more complex entries in the `terminfo` file as of this writing.
Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Lines beginning with "#" are taken as comment lines. Capabilities in terminfo are of three types: boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or particular features, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the fact that the Concept has automatic margins (i.e., an automatic return and linefeed when the end of a line is reached) is indicated by the capability am. Hence the description of the Concept includes am. Numeric capabilities are followed by the character '#' and then the value. Thus cols, which indicates the number of columns the terminal has, gives the value 80 for the Concept. The value 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 given by the two- to five-character capname, an '=' and then a string ending at the next following comma. A delay in milliseconds may appear anywhere in such a capability, enclosed in $<..>$ brackets, as in el=$\texttt{EK}$<3>, and padding characters are supplied by tputs() (see curses(3X)) to provide this delay. The delay can be either a number, e.g., 20, or a number followed by an '=' (i.e., 3*), a '/' (i.e., 5/), or both (i.e., 10*/). A '=' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of lines affected. This is always one unless the terminal has 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 '/' indicates that the padding is mandatory. Otherwise, if the terminal has xon defined, the padding informa-
tion is advisory and will only be used for cost estimates or when the terminal is in raw mode. Mandatory padding will be transmitted regardless of the setting of `xon`.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. Both \E and \e map to an \texttt{ESCAPE} character, \texttt{x} maps to a control–x for any appropriate x, and the sequences \texttt{\n}, \texttt{\l}, \texttt{\r}, \texttt{\t}, \texttt{\b}, \texttt{\f}, and \texttt{\s} give a newline, linefeed, return, tab, backspace, formfeed, and space, respectively. Other escapes include: \texttt{\^} for caret (^); \texttt{\m} for backslash (\); \texttt{\ for comma (,); \ for colon (:); and \texttt{\0} for null. (\texttt{\0} will actually produce \texttt{\200}, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a backslash (e.g., \texttt{\123}).

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second ind in the example above. Note that capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in \texttt{terminfo} and to build up a description gradually, using partial descriptions with \texttt{vi(1)} to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the \texttt{terminfo} file to describe it or the inability of \texttt{vi(1)} to work with that terminal.

To test a new terminal description, set the environment variable \texttt{TERMINFO} to a pathname of a directory containing the compiled description you are working on and programs will look there rather than in \texttt{/usr/lib/terminfo}. To get the padding for insert-line correct (if the terminal manufacturer did not document it) a severe test is to comment out `xon`, edit a large file at 9600 baud with \texttt{vi(1)}, delete 16 or so lines from the middle of the screen, 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.

Basic Capabilities

The number of columns on each line for the terminal is given by the \texttt{cols} numeric capability. If the terminal has a screen, then the number of lines on the screen is given by the \texttt{lines} capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the \texttt{am} capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the \texttt{clear} string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the \texttt{os} capability. If the terminal is a printing terminal, with no soft copy unit, give it both \texttt{hc} and \texttt{os}. (\texttt{os} applies to storage scope terminals, such as Tektronix 4010 series, as well as hard-copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as \texttt{cr}. (Normally this will be carriage return, control M.) If there is a code to produce an audible signal (bell, beep, etc) give this as \texttt{bel}. If the terminal uses the \texttt{xon-xoff} flow-control protocol, like most terminals, specify \texttt{xon}.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as \texttt{cub1}. Similarly, codes to move to the right,
up, and down should be given as `cufl`, `cuu1`, and `cudl`. These local cursor motions should 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.

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 given, and should never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send the `ind` (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the `ri` (reverse index) string. The strings `ind` and `ri` are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are `indn` and `rin` which have the same semantics as `ind` and `ri` except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The `am` capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a `cuf1` from the last column. The only local motion which is defined from the left edge is if `bw` is given, then a `cub1` from the left edge will move to the right edge of the previous row. If `bw` is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch selectable automatic margins, the `terminfo` file usually assumes that this is on; i.e., `am`. If the terminal has a command which moves to the first column of the next line, that command can be given as `nel` (newline). It does not matter if the command clears the remainder of the current line, so if the terminal has no `cr` and `If` it may still be possible to craft a working `nel` out of one or both of them.

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

```
33|tty33|tty|model 33 teletype, bel='G, cols#72, cr='M, cud1=']}, hc, 
```

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,
```

### Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with `printf(3S)`-like escapes (%x) in it. For example, to address the cursor, the `cup` capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by `mrcup`.

The parameter mechanism uses a stack and special `sped` codes to manipulate it in the manner of a Reverse Polish Notation (postfix) calculator. Typically a sequence will push one of the parameters onto the stack and then print it in
some format. Often more complex operations are necessary. Binary operations
are in postfix form with the operands in the usual order. That is, to get $x-5$ one
would use $\%gx\%[S]%-$. 

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() op pop())
- `%%& %%%`%
  bit operations: push(pop() op pop())
- `%== %> %<`
  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 $t b %e c _c$ $t b _c %e c _e c _t b _e c _e b _e b _v %;`
  $c_1$ are conditions, $b_1$ are bodies.

If the "-" flag is used with "[doXs]", then a colon (:) must be placed between
the "%" and the "−" to differentiate the flag from the binary "−" operator,
.e.g. "%:-16.16s".

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12,
needs to be sent \E&aI2c03Y 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%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,
"cup=\T%p1%c%p2%c". Terminals which use "%c" need to be able to back­
space the cursor (cubi), and to move the cursor up one line on the screen
(cuu1). This is necessary because it is not always safe to transmit \n, \D, and \r,
as the system may change or discard them. (The library routines dealing with
terminfo set tty modes so that tabs are never expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus \texttt{cup=\textbackslash E=\%p1\%s\%c\%p2\%s\%c}”. After sending \texttt{\textbackslash E=”}, this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

Cursor Motions
If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as \texttt{home}; similarly a fast way of getting to the lower left-hand corner can be given as \texttt{ill}; this may involve going up with \texttt{cuu} from the home position, but a program should never do this itself (unless \texttt{ill} does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the \texttt{\textbackslash E H} sequence on Hewlett-Packard terminals cannot be used for \texttt{home} without losing some of the other features on the terminal.)

If the terminal has row or column absolute-cursor addressing, these can be given as single parameter capabilities \texttt{hpa} (horizontal position absolute) and \texttt{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 \texttt{cup}. If there are parameterized local motions (e.g., move \textit{n} spaces to the right) these can be given as \texttt{cud}, \texttt{cub}, \texttt{cuf}, and \texttt{cuu} with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have \texttt{cup}, such as the Tektronix 4025.

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 \texttt{el}. If the terminal can clear from the beginning of the line to the current position inclusive, leaving the cursor where it is, this should be given as \texttt{ell}. If the terminal can clear from the current position to the end of the display, then this should be given as \texttt{ed}. \texttt{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 \texttt{ed} is not available.)

Insert/delete line
If the terminal can open a new blank line before the line where the cursor is, this should be given as \texttt{ill}; 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 \texttt{dll}; this is done only from the first position on the line to be deleted. Versions of \texttt{ill} and \texttt{dll} which take a single parameter and insert or delete that many lines can be given as \texttt{il} and \texttt{dl}.

If the terminal has a settable destructive scrolling region (like the VT100) the command to set this can be described with the \texttt{csr} capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command -- the \texttt{sc} and \texttt{rc} (save and restore
cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using ri or ind on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

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 (dll) 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 dll 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 dll all simulate destructive scrolling.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string wind. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the da capability should be given; if display memory can be retained below, then db should be given. These indicate that deleting a line or scrolling a full screen may bring non-blank lines up from below or that scrolling back with ri may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character operations which can be described using terminfo. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type "abc def" using local cursor motions (not spaces) between the abc and the def. Then position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the abc shifts over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability in, which stands for "insert null". While these are two logically separate attributes (one line 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 which have an insert mode and terminals which send a simple sequence to open a blank position on the current line. Give as smir the sequence to get into insert mode. Give as rmir the sequence to leave insert mode. Now give as 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 which 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 will be used. The ich capability, with one parameter, n, will repeat the effects of ichl n times.

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 will affect only speed. Some terminals (notably Datamedia’s) must not have mir because of the way their insert mode works.

Finally, you can specify dchl 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 dchl to work).

A command to erase n characters (equivalent to outputting n blanks without moving the cursor) can be given as ech with one parameter.

**Highlighting, Underlining, and Visible Bells**

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as standout mode (see curses(3X)), representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse-video plus half-bright is good, or reverse-video alone; however, different users have different preferences on different terminals.) The sequences to enter and exit standout mode are given as smso and rmso, respectively. Codes to begin underlining and end underlining can be given as smul and rmul respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Micro-Term MIME, this can be given as uc.

Other capabilities to enter various highlighting modes include blink (blinking), bold (bold or extra-bright), dim (dim or half-bright), invis (blanking or invisible text), prot (protected), rev (reverse-video), sgr0 (turn off all attribute modes), smacs (enter alternate-character-set mode), and rmacs (exit alternate-character-set mode). Turning on any of these modes singly may or may not turn off other modes. If a command is necessary before alternate character set mode is entered, give the sequence in enacs (enable alternate-character-set mode).
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 be supported by sgr, only those for which corresponding separate attribute commands exist. (See the example at the end of this section.)

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), then this can be given 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 the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as smcup and rmcup. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the Tektronix 4025, where smcup sets the command character to be the one used by terminfo. If the smcup sequence will not restore the screen after an rmcup sequence is output (to the state prior to outputting rmcup), specify nrrmc.

If your terminal generates underlined characters by using the underline character (with no special codes needed) even though it does not otherwise overstrike characters, then you should give the capability ul. For terminals where a character overstriking another leaves both characters on the screen, give the capability os. If overstrikes are erasable with a blank, then this should be indicated by giving eo.

Example of highlighting: assume that the terminal under question needs the following escape sequences to turn on various modes.
tparm 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)

Note that each escape sequence requires a 0 to turn off other modes before turning on its own mode. Also note that, as suggested above, standout is set up to be the combination of reverse and dim. Also, since 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 when different sequences are output. 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:

<table>
<thead>
<tr>
<th>sequence</th>
<th>when to output</th>
<th>terminfo translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\E[0</td>
<td>always</td>
<td>\E[0</td>
</tr>
<tr>
<td>;3</td>
<td>if p2 or p6</td>
<td>%?2%p6%t;3%;</td>
</tr>
<tr>
<td>;4</td>
<td>if p1 or p3 or p6</td>
<td>%?1%p3%6%t;4%;</td>
</tr>
<tr>
<td>;5</td>
<td>if p4</td>
<td>%?4%t;5%;</td>
</tr>
<tr>
<td>;7</td>
<td>if p1 or p5</td>
<td>%?1%5%t;7%;</td>
</tr>
<tr>
<td>;8</td>
<td>if p7</td>
<td>%?7%t;8%;</td>
</tr>
<tr>
<td>m</td>
<td>always</td>
<td>m</td>
</tr>
<tr>
<td>\N or \O</td>
<td>if p9 \N, else \O</td>
<td>%?9%t\N%e\O%;</td>
</tr>
</tbody>
</table>

Putting this all together into the sgr sequence gives:

sgr=\E[0%?2%p6%t;3%;?1%3%6%t;4%;?5%t;5%;?1%5%t;7%;?7%?7%t;8%;m?9%t\N%e\O%;;

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 Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as smkx and rmkx. Otherwise the keypad is assumed to always transmit.
The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as kcu<1, kcu<1, kcu<1, kcu<1, and khome respectively. If there are function keys such as f0, f1, ..., f63, the codes they send can be given as kf0, kf1, ..., kf63. If the first 11 keys have labels other than the default £0 through f10, the labels can be given as kf0, kf1, ..., kf10. The codes transmitted by certain other special keys can be given: kll (home down), kbs (backspace), kdc (clear screen or erase key), kdbc (clear the tab stop in this column), kct (clear all tabs), kct (clear the tab stop in this column), kct (clear the tab stop in this column), kct (clear the tab stop in this column), kct (clear the 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 kal, ka3, kb2, kc1, and ke3. 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 given as pfkey, pfloe, and pfx. A string to program their soft-screen labels can be given as pln. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The difference between the capabilities is that pfkey causes pressing the given key to be the same as the user typing the given string; pfloe causes the string to be executed by the terminal; and pfx causes the string to be transmitted to the computer. The capabilities nlab, lw and lh define how many soft labels there are 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.

Tabs and Initialization
If the terminal has hardware tabs, the command to advance to the next tab stop can be given as ht (usually control I). A “backtab” command which moves leftward to the next tab stop can be given as cbt. By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use ht or cbt even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs which are initially set every n spaces when the terminal is powered up, the numeric parameter it is given, showing the number of spaces the tabs are set to. This is normally used by tput init (see tput(l)) to determine whether to set the mode for hardware tab expansion and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the terminfo description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as tbe (dear 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 terminal; iprog, the path name of a program to be run to initialize the terminal; and if, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the terminfo description. They must be sent to the terminal 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 \texttt{mgc}, \texttt{smgl} and \texttt{smgr}; set the tabs using \texttt{tbc} and \texttt{hts}; print the file \texttt{if}; and finally output \texttt{is3}. This is usually done using the \texttt{init} option of \texttt{tput(1)}; see \texttt{profile(4)}.

Most initialization is done with \texttt{is2}. Special terminal modes can be set up without duplicating strings by putting the common sequences in \texttt{is2} and special cases in \texttt{is1} and \texttt{is3}. Sequences that do a harder reset from a totally unknown state can be given as \texttt{rs1}, \texttt{rs2}, \texttt{rf}, and \texttt{rs3}, analogous to \texttt{is1}, \texttt{is2}, \texttt{is3}, and \texttt{if}. (The method using files, \texttt{if} and \texttt{rf}, is used for a few terminals, from \texttt{/usr/lib/tabset/*}; however, the recommended method is to use the initialization and reset strings.) These strings are output by \texttt{tput reset}, which is used when the terminal gets into a wedged state. Commands are normally placed in \texttt{rs1}, \texttt{rs2}, \texttt{rs3}, and \texttt{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 \texttt{is2}, but on some terminals it causes an annoying glitch on the screen and is not normally needed since 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 \texttt{tbc} and \texttt{hts}, the sequence can be placed in \texttt{is2} or \texttt{if}.

If there are commands to set and clear margins, they can be given as \texttt{mgc} (clear all margins), \texttt{smgl} (set left margin), and \texttt{smgr} (set right margin).

Delays

Certain capabilities control padding in the \texttt{tty(7)} driver. These are primarily needed by hard-copy terminals, and are used by \texttt{tput init} to set tty modes appropriately. Delays embedded in the capabilities \texttt{cr}, \texttt{ind}, \texttt{cub1}, \texttt{ff}, and \texttt{tab} can be used to set the appropriate delay bits to be set in the tty driver. If \texttt{pb} (padding baud rate) is given, these values can be ignored at baud rates below the value of \texttt{pb}.

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 \texttt{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 \texttt{tsl} and \texttt{fsl}. (\texttt{fsl} must leave the cursor position in the same place it was before \texttt{tsl}. If necessary, the \texttt{sc} and \texttt{rc} strings can be included in \texttt{tsl} and \texttt{fsl} to get this effect.) The capability \texttt{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 \texttt{tab}, work while in the status line, the flag \texttt{eslok} can be given. A string which turns off the status line (or otherwise erases its contents) should be given as \texttt{dsl}. If the terminal has commands to save and restore the position of the cursor, give them as \texttt{sc} and \texttt{rc}. The status line is normally assumed to be the same width as the rest of the screen, e.g., \texttt{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 \texttt{wsl}.
Line Graphics

If the terminal has a line drawing alternate character set, the mapping of glyph
to character would be given in aasc. 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.

<table>
<thead>
<tr>
<th>glyph name</th>
<th>vt100+ character</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrow pointing right</td>
<td>+</td>
</tr>
<tr>
<td>arrow pointing left</td>
<td>,</td>
</tr>
<tr>
<td>arrow pointing down</td>
<td>.</td>
</tr>
<tr>
<td>solid square block</td>
<td>0</td>
</tr>
<tr>
<td>lantern symbol</td>
<td>I</td>
</tr>
<tr>
<td>arrow pointing up</td>
<td>-</td>
</tr>
<tr>
<td>diamond</td>
<td>'</td>
</tr>
<tr>
<td>checker board (stipple)</td>
<td>a</td>
</tr>
<tr>
<td>degree symbol</td>
<td>f</td>
</tr>
<tr>
<td>plus/minus</td>
<td>g</td>
</tr>
<tr>
<td>board of squares</td>
<td>h</td>
</tr>
<tr>
<td>lower right corner</td>
<td>j</td>
</tr>
<tr>
<td>upper right corner</td>
<td>k</td>
</tr>
<tr>
<td>upper left corner</td>
<td>l</td>
</tr>
<tr>
<td>lower left corner</td>
<td>m</td>
</tr>
<tr>
<td>plus</td>
<td>n</td>
</tr>
<tr>
<td>scan line 1</td>
<td>o</td>
</tr>
<tr>
<td>horizontal line</td>
<td>q</td>
</tr>
<tr>
<td>scan line 9</td>
<td>s</td>
</tr>
<tr>
<td>left tee (</td>
<td>-)</td>
</tr>
<tr>
<td>right tee (—)</td>
<td>u</td>
</tr>
<tr>
<td>bottom tee (⊥)</td>
<td>v</td>
</tr>
<tr>
<td>top tee (⊤)</td>
<td>w</td>
</tr>
<tr>
<td>vertical line</td>
<td>x</td>
</tr>
<tr>
<td>bullet</td>
<td>~</td>
</tr>
</tbody>
</table>

The best way to describe a new terminal’s line graphics set is to add a third
column to the above table with the characters for the new terminal that produce
the appropriate glyph when the terminal is in the alternate character set mode.
For example,
glyph name       vt100+  new tty
                  char     char
upper left corner 1       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=IRmFkTjGq\,x.”.

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as pad. Only the first character of the pad string is used. If the terminal 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 control L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string rep. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, tparm(repeat_char, 'x', 10) is the same as xxxxxxxxxx.

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with cmdch. A prototype command character is chosen which is used in all capabilities. This character is given in the cmdch capability to identify it. The following convention is supported on some UNIX 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 UNIX 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 terminal uses xon/xoff handshaking for flow control, give xon. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted. 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, this fact can be indicated with km. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If
strings exist to turn this "meta mode" on and off, they can be given as `snnn` 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 `ln`. A value of `lm#0` indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

Media copy strings which control an auxiliary printer connected to the terminal can be given as `mc0`: print the contents of the screen, `mc4`: turn off the printer, and `mc5`: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. A variation, `mc5p`, takes one parameter, and leaves 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.

**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`. These are not meant to be construed as deficiencies in the terminals; they are just differences between the working model and the actual hardware. They may be unusual devices or, for some reason, do not have all the features of the `terminfo` model implemented.

Terminals which cannot display tilde (‘’) characters, such as certain Hazeltine terminals, should indicate `hz`.

Terminals which 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.

Those 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 instead necessary 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.

**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,

reva, sgra, smula, 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.

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
curses(3X), printf(3S), term(5).
captoinfo(1M), infocmp(1M), tic(1M), tty(7) in the System Administrator’s Reference Manual.
Chapter 10 of the Programmer’s Guide.

WARNING
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)).

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 will cause unexpected problems.

NOTE
The termcap database (from earlier releases of UNIX System V) may not be supplied in future releases.
NAME
timezone — set default system time zone

SYNOPSIS
/etc/TIMEZONE

DESCRIPTION
This file sets and exports the time zone environmental variable TZ.
This file is "dotted" into other files that must know the time zone.

EXAMPLES
/etc/TIMEZONE for the east coast:

```
    #     Time Zone
    TZ=EST5EDT
    export TZ
```

SEE ALSO
ctime(3C), profile(4).
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.

/* 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; /* Process termination status */
    short e_exit; /* Process exit status */
} ut_exit; /* The exit status of a process */
    /* marked as DEAD_PROCESS. */
    /* 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 /* A "getty" process waiting for login */
define USER_PROCESS 7 /* A user process */
define DEAD_PROCESS 8
#define ACCOUNTING 9
#define UTMAXTYPE ACCOUNTING /* Largest legal value of ut_type */
/* Special strings or formats used in the "ut_line" field when */
/* accounting for something other than a process */
/* No string for the ut_line field can be more than 11 chars + */
/* a NULL in length */
#define RUNLVL_MSG "run-level %c"
#define BOOT_MSG "system boot"
#define OTIME_MSG "old time"
#define NTIME_MSG "new time"

FILES
/etc/utmp
/etc/wtmp

SEE ALSO
getut(3C),
NAME
intro – introduction to miscellany

DESCRIPTION
This section describes miscellaneous facilities such as macro packages, character set tables, etc.
NAME

asci - map of ASCII character set

DESCRIPTION

asci is a map of the ASCII character set, giving both octal and hexadecimal
equivalents of each character, to be printed as needed. It contains:

<table>
<thead>
<tr>
<th>000</th>
<th>001</th>
<th>002</th>
<th>003</th>
<th>004</th>
<th>005</th>
<th>006</th>
<th>007</th>
</tr>
</thead>
<tbody>
<tr>
<td>nul</td>
<td>soh</td>
<td>stx</td>
<td>etx</td>
<td>eot</td>
<td>enq</td>
<td>ack</td>
<td>bel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>011</td>
<td>012</td>
<td>013</td>
<td>014</td>
<td>015</td>
<td>016</td>
<td>017</td>
</tr>
<tr>
<td>bs</td>
<td>ht</td>
<td>nl</td>
<td>vt</td>
<td>np</td>
<td>cr</td>
<td>so</td>
<td>si</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>021</td>
<td>022</td>
<td>023</td>
<td>024</td>
<td>025</td>
<td>026</td>
<td>027</td>
</tr>
<tr>
<td>dle</td>
<td>dc1</td>
<td>dc2</td>
<td>dc3</td>
<td>dc4</td>
<td>nak</td>
<td>syn</td>
<td>etb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>030</td>
<td>031</td>
<td>032</td>
<td>033</td>
<td>034</td>
<td>035</td>
<td>036</td>
<td>037</td>
</tr>
<tr>
<td>can</td>
<td>em</td>
<td>sub</td>
<td>esc</td>
<td>fs</td>
<td>gs</td>
<td>rs</td>
<td>us</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>040</td>
<td>041</td>
<td>042</td>
<td>043</td>
<td>044</td>
<td>045</td>
<td>046</td>
<td>047</td>
</tr>
<tr>
<td>sp</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>050</td>
<td>(</td>
<td>)</td>
<td>+</td>
<td>,</td>
<td>.</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>060</td>
<td>061</td>
<td>062</td>
<td>063</td>
<td>064</td>
<td>065</td>
<td>066</td>
<td>067</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>070</td>
<td>071</td>
<td>072</td>
<td>073</td>
<td>074</td>
<td>075</td>
<td>076</td>
<td>077</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td>105</td>
<td>106</td>
<td>107</td>
</tr>
<tr>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>111</td>
<td>112</td>
<td>113</td>
<td>114</td>
<td>115</td>
<td>116</td>
<td>117</td>
</tr>
<tr>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>131</td>
<td>132</td>
<td>133</td>
<td>134</td>
<td>135</td>
<td>136</td>
<td>137</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>141</td>
<td>142</td>
<td>143</td>
<td>144</td>
<td>145</td>
<td>146</td>
<td>147</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>151</td>
<td>152</td>
<td>153</td>
<td>154</td>
<td>155</td>
<td>156</td>
<td>157</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>161</td>
<td>162</td>
<td>163</td>
<td>164</td>
<td>165</td>
<td>166</td>
<td>167</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>171</td>
<td>172</td>
<td>173</td>
<td>174</td>
<td>175</td>
<td>176</td>
<td>177</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NAME
environ — user environment

DESCRIPTION
An array of strings called the "environment" is made available by exec(2) when
a process begins. By convention, these strings have the form "name=value".
The following names are used by various commands:

PATH  The sequence of directory prefixes that sh(1), time(1), nice(1), nohup(1),
etc., 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.

HOME  Name of the user's login directory, set by login(1) from the password file
passwd(4).

TERM  The kind of terminal for which output is to be prepared. This informa-
tion is used by commands, such as mm(1) or tplot(1G), which may
exploit special capabilities of that terminal.

TZ    Time zone information. The format is xxxnzzz where xxx is standard
local time zone abbreviation, n is the difference in hours from GMT, and
zzz is the abbreviation for the daylight-saving local time zone, if any;
for example, EST5EDT.

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 ALSO
exec(2).

mm(1) in the DOCUMENTER'S WORKBENCH Software Release 2.0 Technical Dis-
NAME
fentl — file control options

SYNOPSIS
#include <fcntl.h>

DESCRIPTION
The fentl(2) function provides for control over open files. This include file describes requests and arguments to fentl and open(2).

/* Flag values accessible to open(2) and fentl(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 the end) */
#define O_SYNC 020 /* synchronous write option */

/* Flag values accessible only to open(2) */
#define O_CREAT 00400 /* open with file create (uses third open arg)*/
#define O_TRUNC 01000 /* open with truncation */
#define O_EXCL 02000 /* exclusive open */

/* fentl(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_SETLK 7 /* Set file lock and wait */
#define F_CHKFL 8 /* Check legality of file flag changes */

/* file segment locking control structure */
struct flock {
    short f_type;
    short f whence;
    long f_start;
    long f_len; /* if 0 then until EOF */
    short f_sysid; /* returned with F_GETLK*/
    short f_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
fentl(2), open(2).
NAME
jagent — host control of windowing terminal

SYNOPSIS
#include <sys/jioctl.h>

ioctl (cntlfd, JAGENT, &arg)

int cntlfd
struct bagent arg

DESCRIPTION
The ioctl(2) system call, when performed on an xt(7) device with the JAGENT
request, allows a host program to send information to a windowing terminal.

ioctl has three arguments:
cntlfd the xt(7) control channel file descriptor
JAGENT the xt(7) ioctl(2) request to invoke a windowing terminal agent rou-
tine.
arg the address of a bagent structure, defined in <sys/jioctl.h> as fol-
lows:

struct bagent {
    int size; /* size of src in & dest out */
    char *src; /* the source byte string */
    char *dest; /* the destination byte string */
};

The src pointer must be initialized to point to a byte string which is
sent to the windowing terminal. See layers(5) for a list of JAGENT
strings recognized by windowing terminals. Likewise, the dest
pointer must be initialized to the address of a buffer to receive a byte
string returned by the terminal. When ioctl(2) is called, the size argu-
ment must be set to the length of the src string. Upon return, size is
set by ioctl(2) to the length of the destination byte string, dest.

RETURN VALUE
Upon successful completion, the size of the destination byte string is returned.
If an error occurs, −1 is returned.

SEE ALSO
ioctl(2), layers(5), libwindows(3X).
LAYERS(5)  (AT&T Windowing Utilities)  LAYERS(5)

NAME
layers — protocol used between host and windowing terminal under layers(1)

SYNOPSIS
#include <sys/jioctl.h>

DESCRIPTION
layers are asynchronous windows supported by the operating system in a windowing terminal. Communication between the UNIX system processes and terminal processes under layers(1) occurs via multiplexed channels managed by the respective operating systems using a protocol as specified in xtproto(5).

The contents of packets transferring data between a UNIX system process and a layer are asymmetric. Data sent from the UNIX system to a particular terminal process is undifferentiated and it is up to the terminal process to interpret the contents of packets.

Control information for terminal processes is sent via channel 0. Process 0 in the windowing terminal performs the designated functions on behalf of the process connected to the designated channel. These packets take the form:

    command, channel

except for timeout and jagent information which take the form

    command, data...

The commands are the bottom eight bits extracted from the following ioctl(2) codes:

    JBOOT     Prepare to load a new terminal program into the designated layer.
    JTERM     Kill the downloaded layer program, and restore the default window program.
    JTIMO     Set the timeout parameters for the protocol. The data consist of two bytes: the value of the receive timeout in seconds, and the value of the transmit timeout in seconds.
    JTIMOM    Set the timeout parameters for the protocol. The data consist of four bytes in two groups: the value of the receive timeout in milliseconds (the low eight bits followed by the high eight bits) and the value of the transmit timeout (in the same format).

    JZOMBOOT   Like JBOOT, but do not execute the program after loading.
    JAGENT    Send a source byte string to the terminal agent routine and wait for a reply byte string to be returned.

The data are from a bagent structure (see jagent(5)) and consist of a one-byte size field followed by a two-byte agent command code and parameters. Two-byte integers transmitted as part of an agent command are sent with the high-order byte first. The response from the terminal is generally identical to the command packet, with the two
command bytes replaced by the return code: 0 for success, -1 for failure. Note that the routines in the \textit{libwindows(3X)} library all send parameters in an \textit{agentrect} structure. The agent command codes and their parameters are as follows:

- \texttt{A_NEWLAYER} followed by a two-byte channel number and a rectangle structure (four two-byte coordinates).
- \texttt{A_CURRENT} followed by a two-byte channel number.
- \texttt{A_DELETE} followed by a two-byte channel number.
- \texttt{A_TOP} followed by a two-byte channel number.
- \texttt{A_BOTTOM} followed by a two-byte channel number.
- \texttt{A_MOVE} followed by a two-byte channel number and a point to move to (two two-byte coordinates).
- \texttt{A_RESHAPE} followed by a two-byte channel number and the new rectangle (four two-byte coordinates).
- \texttt{A_NEW} followed by a two-byte channel number and a rectangle structure (four two-byte coordinates).
- \texttt{A_EXIT} no parameters needed.
- \texttt{A_ROMVERSION} no parameters needed. The response packet contains the size byte, two-byte return code, two unused bytes, and the parameter part of the terminal id string (e.g., "8;7;3").

Packets from the windowing terminal to the UNIX system all take the following form:

\texttt{command, data...}

The single-byte commands are as follows:

- \texttt{C_SENDCHAR} Send the next byte to the UNIX system process.
- \texttt{C_NEW} Create a new UNIX system process group for this layer. Remember the window size parameters for this layer. The data for this command is in the form described by the \textit{jwinsize} structure. The size of the window is specified by two 2-byte integers, sent low byte first.
- \texttt{C_UNBLK} Unblock transmission to this layer. There is no data for this command.
- \texttt{C_DELETE} Delete the UNIX system process group attached to this layer. There is no data for this command.
- \texttt{C_EXIT} Exit. Kill all UNIX system process groups associated with this terminal and terminate the session. There is no data for this command.
C_DEFUNCT  Layer program has died, send a terminate signal to the UNIX system process groups associated with this terminal. There is no data for this command.

C_SENDNCHARS  The rest of the data are characters to be passed to the UNIX system process.

C_RESHAPE  The layer has been reshaped. Change the window size parameters for this layer. The data takes the same form as for the C_NEW command.

SEE ALSO
libwindows(3X), jagent(5), xtproto(5).
NAME
math — math functions and constants

SYNOPSIS
#include <math.h>

DESCRIPTION
This file contains declarations of all the functions in the Math Library (described in Section 3M), as well as various functions in the C Library (Section 3C) that return floating-point values.

It defines the structure and constants used by the matherr(3M) error-handling mechanisms, including the following constant used as an error-return value:

HUGE The maximum value of a single-precision floating-point number.

The following mathematical constants are defined for user convenience:

M_E The base of natural logarithms (e).
M_LOG2E The base-2 logarithm of e.
M_LOG10E The base-10 logarithm of e.
M_LN2 The natural logarithm of 2.
M_LN10 The natural logarithm of 10.
M_PI The ratio of the circumference of a circle to its diameter.
M_PI_2 M_PI/2.
M_PI_4 M_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 will introduce a mark called name that will be treated the same as a function entry point. Execution of the mark will add to a counter for that mark, and program-counter time spent will be 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 will be ignored.

EXAMPLE
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 cird, 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), expr(1), etc., 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 unpleasantly 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.
This is the abnormal return from the compile routine. The argument val is an error number (see table below for meanings). This call should never return.

<table>
<thead>
<tr>
<th>ERROR</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Range endpoint too large.</td>
</tr>
<tr>
<td>16</td>
<td>Bad number.</td>
</tr>
<tr>
<td>25</td>
<td>&quot;\digit&quot; out of range.</td>
</tr>
<tr>
<td>36</td>
<td>Illegal or missing delimiter.</td>
</tr>
<tr>
<td>41</td>
<td>No remembered search string.</td>
</tr>
<tr>
<td>42</td>
<td>( ) imbalance.</td>
</tr>
<tr>
<td>43</td>
<td>Too many ).</td>
</tr>
<tr>
<td>44</td>
<td>More than 2 numbers given in { }.</td>
</tr>
<tr>
<td>45</td>
<td>} expected after ).</td>
</tr>
<tr>
<td>46</td>
<td>First number exceeds second in { }.</td>
</tr>
<tr>
<td>49</td>
<td>[ ] imbalance.</td>
</tr>
<tr>
<td>50</td>
<td>Regular expression overflow.</td>
</tr>
</tbody>
</table>

The syntax of the compile routine is as follows:

```c
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:

```c
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 `cirmf` 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 `cirmf` should be saved for each compiled expression and `cirmf` 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 `loc1` is equal to the point in the string at sometime during the backing up process, `advance` will break out of the loop that backs up and will return 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/\*//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)`:

```c
#define INIT
#define GETC(sp) (*sp++)
#define PEEKC(sp) (*sp)
#define UNGETC(sp) (--sp)
#define RETURN(sp) return;
#define ERROR(sp) regerr()
```

621
#include <regexp.h>

... (void) compile(argc, expbuf, &expbuf[ESIZE], '\0');

... if (step(linebuf, expbuf))
    succeed();

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;
    ushort st_ino;
    ushort st_mode;
    short  st_nlink;
    ushort st_uid;
    ushort st_gid;
    dev_t  st_rdev;
    off_t  st_size;
    time_t st_atime;
    time_t st_mtime;
    time_t st_ctime;
};

#define  S_IFMT   0170000 /* type of file */
#define  S_IFDIR  0040000 /* directory */
#define  S_IFCHR  0020000 /* character special */
#define  S_IFBLK  0060000 /* block special */
#define  S_IFREG  0100000 /* regular */
#define  S_IFIFO  0010000 /* fifo */
#define  S_ISUID  040000 /* set user id on execution */
#define  S_ISGID  020000 /* set group id on execution */
#define  S_ISVTX  010000 /* save swapped text even after use */
#define  S_IRUSR  004000 /* read permission, owner */
#define  S_IWUSR  002000 /* write permission, owner */
#define  S_IXUSR  001000 /* execute permission, owner */
#define  S_IRGRP  000400 /* read permission, group */
#define  S_IWGRP  000200 /* write permission, group */
#define  S_IXGRP  000100 /* execute permission, group */
#define  S_IROTH  000070 /* read, write, execute: other */
#define  S_IWOTH  000040 /* write permission: other */
#define  S_IXOTH  000010 /* execute permission: other */
#define  S_ISMT   000004 /* read permission: other */
#define S_IWOTH 00002 /* write permission: other */
#define S_IXOTH 00001 /* execute permission: other */

SEE ALSO
stat(2), types(5).
TERM - conventional names for terminals

These names are used by certain commands (e.g., man(1), tabs(1), tput(1), vi(1) and 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 submodels, 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:

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-w</td>
<td>Wide mode (more than 80 columns)</td>
<td>att4425-w</td>
</tr>
<tr>
<td>-am</td>
<td>With auto. margins (usually default)</td>
<td>vt100-am</td>
</tr>
<tr>
<td>-nam</td>
<td>Without automatic margins</td>
<td>vt100-nam</td>
</tr>
<tr>
<td>-n</td>
<td>Number of lines on the screen</td>
<td>aaa-60</td>
</tr>
<tr>
<td>-na</td>
<td>No arrow keys (leave them in local)</td>
<td>c100-na</td>
</tr>
<tr>
<td>-np</td>
<td>Number of pages of memory</td>
<td>c100-4p</td>
</tr>
<tr>
<td>-rv</td>
<td>Reverse video</td>
<td>att4415-rv</td>
</tr>
</tbody>
</table>

To avoid conflicts with the naming conventions used in describing the different modes of a terminal (e.g., -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+n1.

Some of the known terminal names may include the following (for a complete list, type: ls -C /usr/lib/terminfo/?):
2621, hp2621
2631
2631-c
2631-e
2640, hp2640
2645, hp2645
3270
33, tty33
35, tty35
37, tty37
4000a
4014, tek4014
40, tty40
43, tty43
4410, 5410
4410-nfk, 5410-nfk
4410-nsl, 5410-nsl
4410-v, 5410-w
4410v1, 5410v1
4410v1-w, 5410v1-w
4415, 5420
4415-nl, 5420-nl
4415-rv, 5420-rv
4415-rv-nl, 5420-rv-nl
4415-w, 5420-w
4415-w-nl, 5420-w-nl
4415-w-rv, 5420-w-rv
4415-w-rv-nl, 5420-w-rv-nl
4418, 5418
4418-w, 5418-w
4420
4424
4424-2
4425, 5425
4425-fk, 5425-fk
4425-nl, 5425-nl
4425-w, 5425-w
4425-w-fk, 5425-w-fk
4425-nl-w, 5425-nl-w
4426
450
450-12
500, att500
510, 510a

Hewlett-Packard 2621 series
Hewlett-Packard 2631 line printer
Hewlett-Packard 2631 line printer - compressed mode
Hewlett-Packard 2631 line printer - expanded mode
Hewlett-Packard 2640 series
Hewlett-Packard 2645 series
IBM Model 3270
AT&T Teletype Model 33 KSR
AT&T Teletype Model 35 KSR
AT&T Teletype Model 37 KSR
Trendata 4000a
TEKTRONIX 4014
AT&T Teletype Dataspeed 40/2
AT&T Teletype Model 43 KSR
AT&T 4410/5410 terminal in 80-column mode - version 2
AT&T 4410/5410 without function keys - version 1
AT&T 4410/5410 without pln defined
AT&T 4410/5410 in 132-column mode
AT&T 4410/5410 terminal in 80-column mode - version 1
AT&T 4410/5410 terminal in 132-column mode - version 1
AT&T 4410/5410 in 80-column mode
AT&T 4415/5420 without changing labels
AT&T 4415/5420 80 columns in reverse video
AT&T 4415/5420 reverse video without changing labels
AT&T 4415/5420 in 132-column mode
AT&T 4415/5420 in 132-column mode without changing labels
AT&T 4415/5420 132 columns in reverse video
AT&T 4415/5420 132 columns reverse video without changing labels
AT&T 4415/5420 in 80-column mode
AT&T 4415/5420 in 132-column mode
AT&T Teletype Model 4420
AT&T Teletype Model 4424
AT&T Teletype Model 4424 in display function group ii
AT&T 4425/5425
AT&T 4425/5425 without function keys
AT&T 4425/5425 without changing labels in 80-column mode
AT&T 4425/5425 in 132-column mode
AT&T 4425/5425 without function keys in 132-column mode
AT&T 4425/5425 without changing labels in 132-column mode
AT&T Teletype Model 4426S
DASI 450 (same as Diablo 1620)
DASI 450 in 12-pitch mode
AT&T-IS 500 terminal
AT&T 510/510a in 80-column mode
## TERM(5)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>513bct,att513</td>
<td>AT&amp;T 513 bct terminal</td>
</tr>
<tr>
<td>5320</td>
<td>AT&amp;T 5320 hardcopy terminal</td>
</tr>
<tr>
<td>5420_2</td>
<td>AT&amp;T 5420 model 2 in 80-column mode</td>
</tr>
<tr>
<td>5420_2-w</td>
<td>AT&amp;T 5420 model 2 in 132-column mode</td>
</tr>
<tr>
<td>5620,dmd</td>
<td>AT&amp;T 5620 terminal 88 columns</td>
</tr>
<tr>
<td>5620-24,dmd-24</td>
<td>AT&amp;T Teletype Model DMD 5620 in a 24x80 layer</td>
</tr>
<tr>
<td>5620-34,dmd-34</td>
<td>AT&amp;T Teletype Model DMD 5620 in a 34x80 layer</td>
</tr>
<tr>
<td>610,610bct</td>
<td>AT&amp;T 610 bct terminal in 80-column mode</td>
</tr>
<tr>
<td>610-w,610bct-w</td>
<td>AT&amp;T 610 bct terminal in 132-column mode</td>
</tr>
<tr>
<td>7300,pc7300,unix_pc</td>
<td>AT&amp;T UNIX PC Model 7300</td>
</tr>
<tr>
<td>735,ti</td>
<td>Texas Instruments TI735 and TI725</td>
</tr>
<tr>
<td>745</td>
<td>Texas Instruments TI745</td>
</tr>
<tr>
<td>dumb</td>
<td>generic name for terminals that lack reverse line-feed and other special escape sequences</td>
</tr>
<tr>
<td>hp</td>
<td>Hewlett-Packard (same as 2645)</td>
</tr>
<tr>
<td>lp</td>
<td>generic name for a line printer</td>
</tr>
<tr>
<td>pt505</td>
<td>AT&amp;T Personal Terminal 505 (22 lines)</td>
</tr>
<tr>
<td>pt505-24</td>
<td>AT&amp;T Personal Terminal 505 (24-line mode)</td>
</tr>
<tr>
<td>sync</td>
<td>generic name for synchronous Teletype Model 4540-compatible terminals</td>
</tr>
</tbody>
</table>

Commands whose behavior depends on the type of terminal should accept arguments of the form `-Tterm` 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

- `curses(3X), profile(4), terminfo(4), environ(5)`, `man(1), sh(1), stty(1), tabs(1), tput(1), tplot(1G), vi(1)` in the *User’s Reference Manual*.
- `infocomp(1M)` in the *System Administrator’s Reference Manual*.
- Chapter 10 of the *Programmer’s Guide*.

### 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 UNIX system code; some
data of these types are accessible to user code:

typedef struct { int r[]; } *physadr;
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 ushort ino_t;
typedef short cnU;
typedef long time_t;
typedef int label_t[10];
typedef short dev_t;
typedef long off_t;
typedef long paddr_t;
typedef int key_t;
typedef unsigned char use_t;
typedef short sysid_t;
typedef short index_t;
typedef short lock_t;
typedef unsigned int size_t;

The form daddr_t is used for disk addresses except in an i-node on disk, see
fs(4). Times are encoded in seconds since 00:00:00 GMT, 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 begin­
ning 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 (in most implementations, 0x8000).
HIBITL The value of a long integer with only the high-order bit set (in most implementations, 0x80000000).
HIBITI The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).
MAXSHORT The maximum value of a signed short integer (in most implementations, 0x7FFF \equiv 32767).
MAXLONG The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF \equiv 2147483647).
MAXINT The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).

MAXFLOAT, LN_MAXFLOAT The maximum value of a single-precision floating-point number, and its natural logarithm.
MAXDOUBLE, LN_MAXDOUBLE The maximum value of a double-precision floating-point number, and its natural logarithm.
MINFLOAT, LN_MINFLOAT The minimum positive value of a single-precision floating-point number, and its natural logarithm.
MINDOUBLE, LN_MINDOUBLE The minimum positive value of a double-precision floating-point number, and its natural logarithm.

FSIGNIF The number of significant bits in the mantissa of a single-precision floating-point number.

DSIGNIF The number of significant bits in the mantissa of a double-precision floating-point number.

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 will return the next argument in the list pointed to by pvar. Type is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, as it cannot be determined at runtime.
va_end is used to clean up.

Multiple traversals, each bracketed by va_start ... va_end, are possible.

EXAMPLE
This example is a possible implementation of execl(2).

#include <varargs.h>
#define MAXARGS 100

/* execl is called by
   execl(file, arg1, arg2, ..., (char *)0);
*/
excl(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, `exec1` is passed a zero pointer to signal the end of the list. `Printf` can tell how many arguments are there by the format.
It is non-portable to specify a second argument of `char`, `short`, or `float` to `va_arg`, since arguments seen by the called function are not `char`, `short`, or `float`. C converts `char` and `short` arguments to `int` and converts `float` arguments to `double` before passing them to a function.
NAME
   xtpdriver – multiplexed channels protocol used by xt(7) driver

DESCRIPTION
   The xt(7) driver contains routines which implement a multiplexed, multi-
   buffered, full-duplex protocol with guaranteed delivery of ordered data via an
   8-bit byte data stream. This protocol is used for communication between mul-
   tiple UNIX system host processes and an AT&T windowing terminal operating
   under layers(1).

   The protocol uses packets with a 2-byte header containing a 3-bit sequence
   number, 3-bit channel number, control flag, and data size. The data part of a
   packet may not be larger than 32 bytes. The trailer contains a CRC-16 code in 2
   bytes. Each channel is double-buffered.

   Correctly received packets in sequence are acknowledged with a control packet
   containing an ACK; however, out of sequence packets generate a control packet
   containing a NAK, which will cause the retransmission in sequence of all unack-
   nowledged packets.

   Unacknowledged packets are retransmitted after a timeout interval which is
   dependent on baud rate. Another timeout parameter specifies the interval after
   which incomplete receive packets are discarded.

FILES
   /usr/include/sys/xtpproto.h    channel multiplexing protocol definitions

SEE ALSO
   layers(5).
Other books in the Prentice-Hall C and UNIX® Systems Library

- The C Programmer’s Handbook  
  Bell Labs/M. I. Bolsky
- The UNIX System User’s Handbook  
  Bell Labs/M. I. Bolsky
- The Vi User’s Handbook  
  Bell Labs/M. I. Bolsky
- UNIX System Software Readings  
  AT&T UNIX PACIFIC
- UNIX System Readings and Applications, Volume I  
  Bell Labs
- UNIX System Readings and Applications, Volume II  
  Bell Labs
- UNIX System V Utilities Release Notes  
  AT&T
- UNIX System V Streams Primer  
  AT&T
  AT&T
  AT&T
  AT&T
- UNIX System V Streams Programmer’s Guide  
  AT&T
- UNIX System V Network Programmer’s Guide  
  AT&T
- UNIX System V Programmer’s Guide  
  AT&T