



Burroughs

XE 500 CENTIX™

Operations
Reference
Manual

Volume 4: System
Operations, Part 2

Relative To Release Level 6.0
Priced Item
November 1986

Distribution Code SA
Printed in U S America
1207891



Burroughs

XE 500 CENTIX™

Operations Reference Manual

Copyright © 1986, Burroughs Corporation, Detroit, Michigan 48232

™Trademark of Burroughs Corporation

Volume 4: System Operations, Part 2

Relative To Release Level 6.0
Priced Item
November 1986

Distribution Code SA
Printed in U S America
1207891

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THIS DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Burroughs, if any, with respect to the products described in this document are set forth in such License or Agreement. Burroughs cannot accept any financial or other responsibility that may be the result of your use of the information or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded, using the Product Improvement Card at the back of this manual, or remarks may be addressed directly to Burroughs Corporation, Corporate Product Information East, 209 W. Lancaster Ave., Paoli, PA 19301, U.S.A.

About This Manual

Purpose

The purpose of the *XE 500 CENTIX Operations Reference Manual* is to provide a comprehensive reference for the XE 500 CENTIX operating system.

Scope

This manual describes the commands, system calls, libraries, data files, and device interfaces that make up the CENTIX Operating System running on the XE 500 computer.

Audience

Volumes 1 and 2 of this manual are intended for all users of the CENTIX operating system. CENTIX system programmers are the primary audience for Volumes 3 and 4.

Prerequisites

General users of the CENTIX system should be familiar with the particular environments in which they will be working. A section called **Getting Started**, preceding the Shell Command descriptions in Volumes 1 and 2, provides a generic CENTIX tutorial.

Programmers should have an understanding of the CENTIX operating system structure and should be experienced at writing programs in the C programming language.

How to Use This Manual

Use this manual as a starting point to find the documentation for a CENTIX feature with which you are unfamiliar. To find the entry you need, refer to the following:

- **Permuted Index.** This indexes each significant word in each entry's description. A complete Permuted Index for the whole manual is in each volume.
- **Contents Listing.** Included in the Contents Listing is an alphabetical list of entries, under the appropriate sections, together with the entry descriptions. Each volume contains the Contents Listing.
- **Related Shell Command Entries.** This section, for Volumes 1 and 2 only, groups together related shell command entries that are in Section 1.

Organization

This manual consists of six sections:

Section 1, **Shell Commands**, describes programs that are intended to be invoked directly by the user through the CENTIX System shell.

Section 2, **System Calls**, describes the entries into the CENTIX kernel, including the C language interfaces.

Section 3, **Library Functions**, describes the available library functions and subroutines.

Section 4, **Special File Formats**, documents the structure of particular kinds of files.

Section 5, **Miscellaneous Facilities**, includes descriptions of macro packages, character set tables, and so on.

Section 6, **Device Files**, describes various device files that refer to specific hardware peripherals and CENTIX System device drivers.

Related Product Information

XE 500 CENTIX Administration Guide

XE 500 CENTIX centrEASE Operations Reference Manual

XE 500 CENTIX C Language Programming Reference Manual

XE 500 CENTIX Programming Guide

XE 500 CENTIX Operations Guide

Contents

Volume 1: Shell Operations, Part 1

Section 1: Shell Commands 1-1

intro	introduction to shell commands
accept	allow LP requests
adb	absolute debugger
admin	create and administer SCCS files
allrc	system initialization shell script
apnum	print Application Processor number
ar	archive and library maintainer for portable object code archives
as	mc68010 assembler
at, batch	execute commands at a later time
awk	pattern scanning and processing language
banner	make posters
basename	deliver portions of path names
batch	execute commands at a later time
bc	high-precision arithmetic language
bcheckrc	system initialization shell script
bcopy	interactive block copy
bdiff	big diff
bfs	big file scanner
brc	system initialization shell script
cal	print calendar
calendar	reminder service
cancel	cancel requests to an LP line printer
cat	concatenate and print files
cb	C program beautifier
cc	C compiler
cd	change working directory
cdc	change the delta commentary of an SCCS delta

centreCAP	function key shell for unskilled users
centreWINDOW	window management
cflow	generate C flow graph
chgrp	change group
chmod	change mode
chown	change owner
chroot	change root directory for a command
clear	clear terminal screen
cli	clear inode
cmp	compare two files
col	filter reverse line-feeds
comb	combine SCCS deltas
comm	select or reject lines common to two sorted files
conrc	system initialization shell script
console	control Application Processor pseudoconsole
convert	convert object and archive files to common formats
cp	copy files
cpio	copy file archives in and out
cpp	the C language preprocessor
cpset	install object files in binary directories
cron	clock daemon
crontab	user crontab file
crup	create file system partition
csplit	context split
ct	spawn getty to a remote terminal
ctrace	C program debugger
cu	call another computer system
cut	cut out selected fields of each line of a file
cxref	generate C program cross reference
date	print and set the date
dc	desk calculator

dcopy	copy file systems for optimal access time
dd	convert and copy a file
delta	make a delta (change) to an SCCS file
devnm	device name
df	report number of free disk blocks
diff	differential file comparator
diff3	3-way differential file comparison
dircmp	directory comparison
dirname	deliver portions of path names
disable	disable LP printers
du	summarize disk usage
dump	dump selected parts of an object file
echo	echo arguments
ed, red	text editor
edit	text editor
egrep	search a file for a pattern
enable	enable LP printers
env	set environment for command execution
ex, edit	text editor
expr	evaluate arguments as an expression
factor	factor a number
false	false
ff	list file names and statistics for a file system
fgrep	search a file for a pattern
file	determine file type
finc	fast incremental backup
find	find files
fold	fold long lines for finite width output device
fpsar	File Processor system activity reporter
frec	recover files from a backup tape

fsck	file system consistency check and interactive repair
fsdb	file system debugger
fwtmp	manipulate connect accounting records
get	get a version of an SCCS file
getopt	parse command options
getty	set terminal type, modes, speed, and line discipline
grep	search a file for a pattern
grpck	group file checker
gtdl	RS-232-C terminal download
halt	terminate all processing
hd	hexadecimal and ASCII file dump
head	give first few lines
help	ask for help for SCCS commands
hyphen	find hyphenated words
icode	process control initialization
id	print user and group IDs and names
init	process control initialization
install	install commands
ipcrm	remove a message queue, semaphore set or shared memory id
ipcs	report inter-process communication facilities status
join	relational database operator
keystate	print XE 550 front panel keyswitch setting
kill	terminate a process
killall	kill all active processes
labelit	file system label checking
ld	link editor for common object files
lex	generate programs for simple lexical tasks
line	read one line
link	exercise link and unlink system calls
lint	a C program checker

ln	link files
login	sign on
logname	get login name
lorder	find ordering relation for an object library
lp	send requests to an LP line printer
lpadmin	configure the LP spooling system
lpmove	move LP requests
lpr	line printer spooler
lpsched	start the LP request scheduler
lpset	set parallel line printer options
lpshut	stop the LP request scheduler
lpstat	print LP status information
ls	list contents of directories

Volume 2: Shell Operations, Part 2

Section 1: Shell Commands (Cont.) 1-283

m4	macro processor
machid	mc68k, pdp11, u3b, vax, iAPX286 - processor type
mail	send or read mail
make	maintain, update, and regenerate groups of programs
mesg	permit or deny messages
mkboot	reformat CENTIX kernel and copy it to BTOS
mkdir	make a directory
mkfs	construct a file system
mklost+found	make a lost+found directory for fsck
mknod	build special file
more	text perusal
mount	mount and dismount file system
mv	move files
mvdir	move a directory

mvtpy	move PT/GT local printer device files
ncheck	generate names from i-numbers
newform	change the format of a text file
newgrp	log in to a new group
news	print news items
nice	run a command at low priority
nl	line numbering filter
nm	print name list of common object file
nohup	run a command immune to hangups and quits
od	octal dump
ofcli	command line interpreter for interactive BTOS JCL
ofcopy	copy to or from the BTOS file system
ofed	edit BTOS files
ofls	list BTOS files and directories
ofvi	edit BTOS files
pack	compress and expand files
page	text perusal
passwd	change login password
paste	merge same lines of several files or subsequent lines of one file
path	locate executable file for command
pbuf	print the kernel print buffer
perc	describe BTOS error return code (erc)
pg	file perusal filter for soft-copy terminals
pmon	display statistics for an Application Processor
pr	print files
prfdc	operating system profiler
prfld	operating system profiler
prfpr	operating system profiler
prfsnap	operating system profiler
prfstat	operating system profiler

prof	display profile data
profiler	operating system profiler
prs	print an SCCS file
ps	report process status
pstat	ICC statistics for processor
ptdl	RS-232-C terminal download
ptx	permuted index
pwck	password file checker
pwd	working directory name
rc	system initialization shell script
red	restricted version text editor
regcmp	regular expression compiler
reject	prevent LP requests
renice	alter priority of running process by changing nice
rm	remove files
rmdel	remove a delta from an SCCS file
rmdir	remove directories
rsh	shell, restricted command programming language
sa1	system activity reporter
sa2	system activity reporter
sact	print current SCCS file editing activity
sadc	system activity reporter
sadp	disk access profiler
sag	system activity graph
sar	system activity reporter
sarpkg	system activity report package
sccsdiff	compare two versions of an SCCS file
script	make typescript of terminal session
sdb	symbolic debugger
sdiff	side-by-side difference program

sed	stream editor
setmnt	establish mount table
setuname	set name of system
sh	shell, the standard/restricted command programming language
shutdown	terminate all processing
size	print section sizes of common object files
sleep	suspend execution for an interval
sort	sort and/or merge files
spawn	execute a process on a specific Application Processor
spawnsrv	service spawn execution requests
spell	hashmake, spellin, hashcheck - find spelling errors
split	split a file into pieces
strip	strip symbol and line number information from a common object file
stty	set the options for a terminal
su	become super-user or another user
sum	print checksum and block count of a file
sync	update the super block
tabs	set tabs on a terminal
tail	deliver the last part of a file
tar	tape file archiver
tdl	RS-232-C terminal download
tee	pipe fitting
telinit	process control initialization
test	condition evaluation command
tic	terminfo compiler
tidc	display decompiled version of terminfo entry
time	time a command
timex	time a command; report process data and system activity
touch	update access and modification times of a file
tput	query terminfo data base

tr	translate characters
true	provide truth values
tset	set terminal, terminal interface, and terminal environment
tsort	topological sort
tty	get the terminal's name
umask	set file-creation mode mask
umount	dismount file system
uname	print name of system
unset	undo a previous get of an SCCS file
uniq	report repeated lines in a file
units	conversion program
update	provide disk synchronization
uuclean	uucp spool directory clean-up
uucp	copy files between computer systems
uulog	query a summary log of uucp and uux transactions
uuname	list uucp names of known systems
uupick	accept or reject files transmitted by uuto
uustat	uucp status inquiry and job control
uusub	monitor uucp network
uuto	public computer system-to-computer system file copy
uux	computer system to computer system command execution
val	validate SCCS file
vc	version control
vi	screen-oriented (visual) display editor
view	visual editor
volcopy	copy file systems with label checking
wait	await completion of process
wall	write to all users
wc	word count
what	identify SCCS files

who	who is on the system
whodo	who is doing what
wm	window management
write	write to another user
wtmpfix	manipulate connect accounting records
xargs	construct argument list(s) and execute command
yacc	yet another compiler-compiler

Volume 3: System Operations, Part 1

Section 2: System Calls 2-1

intro	introduction to system calls and error numbers
access	determines the accessibility of a file
acct	enable or disable process accounting
alarm	set a process alarm clock
brk	change data segment spaced allocation
chdir	changes the current working directory
chmod	change mode of file
chown	changes the owner and/or group of a file
chroot	change the root directory
close	close a file descriptor
creat	create a new file or rewrite an existing one
dup	duplicate an open file descriptor
exAllocExch	allocate exchange
exCall	send a request and wait for the response
exchanges	obtain and abandon exchanges
exCheck	examine an ICC message queue
exCnxSendOnDealloc	make final requests
exCpRequest	remove a request from an exchange
exCpResponse	remove a response from an exchange
exDeallocExch	deallocate exchange

exDiscard	remove a response from an exchange
exec	execute files
execl	execute files
execle	execute a file
execlp	execute a file
execv	execute a file
execve	execute a file
execvp	execute a file
exfinal	make final requests
exit	terminate process
exReject	remove a request from an exchange
exRequest	send a message to a server
exRespond	send a message to a client
exSendOnDealloc	make final requests
exServeRq	appropriate a request code
exWait	examine an ICC message queue
fcntl	file control
fork	create a new process
fstat	get file status
getegid	get effective group ID
geteuid	get effective user ID
getgid	get real group ID
getpgrp	get process group ID
getpid	get process, process group, and parent process IDs
getppid	get parent process ID
getuid	get real user, effective user, real group, and effective group IDs
ioctl	control device
kill	send a signal to a process or a group of processes
link	link to a file
locking	exclusive access to regions of a file

lseek	move read/write file pointer
mkdir	makes a directory, or a special or ordinary file
mount	mount a file system
msgctl	message control operations
msgget	get message queue
msgop	message operations
nice	change priority of a process
open	open a file for reading or writing
pause	suspend process until signal
pipe	create an interprocess channel
plock	lock process, text, or data in memory
profil	execution time profile
ptrace	process trace
read	read from a file
sbrk	change data segment space allocation
semctl	semaphore control operations
semget	get set of semaphores
semop	semaphore operations
setgid	get group ID
setpgrp	set process group ID
setuid	set user ID
shmctl	shared memory control operations
shmget	get shared memory segment
shmop	shared memory operations
signal	specify what to do upon receipt of a signal
stat	get file status
stime	set time
swrite	synchronous write on a file
sync	update super-block
syslocal	special system requests

time	get time
times	get process and child process times
ulimit	get and set user limits
umask	set and get the file creation mask
umount	unmount a file system
uname	get name of current CENTIX system
unlink	remove directory entry
ustat	get file system statistics
utime	set file access and modification times
wait	wait for a child process to stop or terminate
write	write on a file

Section 3: Library Functions 3-1

intro	introduction to libraries and subroutines
a64i	convert between long integer and base-64 ASCII string
abort	generate an IOT fault
abs	return integer absolute value
assert	verify program assertion
atof	convert ASCII string to floating-point number
Bessel	Bessel functions
bsearch	binary search a sorted table
clock	report CPU time used
conv	translate characters
crypt	generate DES encryption
ctermid	generate file name for terminal
ctime	convert date and time to string
ctype	classify characters
courses	CRT screen handling and optimization package
userid	get character login name of the user
dial	establish and release an out-going terminal line connection

drand48	generate uniformly distributed pseudo-random numbers
ecvt	convert floating-point number to string
end	last locations in programs
erf	error function and complementary error function
exp	exponential, logarithm, power, square root functions
fclose	close or flush a stream
ferror	stream status inquiries
floor	floor, ceiling, remainder, absolute value functions
fopen	open a stream
fread	binary input/output
frexp	manipulate parts of floating-point numbers
fseek	reposition a file pointer in a stream
ftw	walk a file tree
gamma	log gamma function
getc	get character or word from a stream
getcwd	get the path-name of the current working directory
getenv	return value for environment name
getgrent	get group file entry
getlogin	get login name
getopt	get option letter from argument vector
getpass	read a password
getpw	get name from UID
getpwent	get password file entry
gets	get a string from a stream
getut	access utmp file entry
hsearch	manage hash search tables
hypot	Euclidean distance function
l3tol	convert between 3-byte integers and long integers
ldahread	read the archive header of a member of an archive file
ldclose	close a common object file

ldfthead	read the file header of a common object file
ldgetname	retrieve symbol name for common object file symbol table entry
ldlread	manipulate line number entries of a common object file function
ldlseek	seek to line number entries of a section of a common object file
ldohseek	seek to the optional file header of a common object file
ldopen	open a common object file for reading
ldrseek	seek to relocation entries of a section of a common object file
ldshread	read an indexed/named section header of a common object file
ldsseek	seek to an indexed/named section of a common object file
ldtbindex	compute the index of a symbol table entry of a common object file
ldtbread	read an indexed symbol table entry of a common object file
ldtbseek	seek to the symbol tsble of a common object file
lockf	record locking on files
logname	return login name of user
lsearch	linear search and update
malloc (fast version)	fast main memory allocator
malloc	main memory allocator
matherr	error-handling function
memory	memory operations
mktemp	make a unique file name
monitor	prepare execution profile
nlist	get entries from the name list
ocurse	optimized screen functions
ofCreate	allocate BTOS files
ofDir	BTOS directory functions
ofOpenFile	access BTOS files
ofRead	input/output on a BTOS file
ofRename	rename a BTOS file
ofStatus	BTOS file status
perror	system error messages

popen	initiate pipe to/from a process
printf	print formatted output
putc	put character or word on a stream
putenv	change or add value to environment
putpwent	write password file entry
puts	put a string on a stream
qsort	quicker sort
quAdd	add a new entry to a BTOS queue
quRead	examine BTOS queue
quRemove	take back a BTOS queue request
rand	simple random number generator
regcmp	compile and execute regular expression
scanf	convert formatted input
setbuf	assign buffering to a stream
setjmp	non-local goto
sinh	hyperbolic functions
sleep	suspend execution for interval
spawn	execute a process on a specific Application Processor
sputl	access long integer data in a machine-dependent fashion
spwait	wait for a spawned process to terminate
ssignal	software signals
stdio	standard buffered input/output package
stdipc	standard interprocess communication package (ftok)
string	string operations
strtod	convert string to double-precision number
strtol	convert string to integer
swab	swap bytes
swapshort	translate byte orders to Motorola/Intel
system	issue a shell command
termcap	terminal independent operations

tmpfile	create a temporary file
tmpnam	create a name for a temporary file
trig	trigonometric functions
tsearch	manage binary search trees
ttyname	find name of a terminal
ttyslot	find the slot in the utmp file of the current user
ungetc	push character back into input stream
vprintf	print formatted output of a varargs argument list
wmgetid	get window ID
wmlayout	get terminal's window layout
wmop	window management operations
wmsetid	associate a file descriptor with a window

Volume 4: System Operations, Part 2

Section 4: Special File Formats 4-1

intro	introduction to special file formats
a.out	common assembler and link editor output
ar	common archive file format
checklist	list of file systems processed by fsck
core	format of core image file
cpio	format of cpio archive
dir	format of directories
filehdr	file header for common object file
fs	format of file system
fspec	format specification in text file
gettydefs	speed and terminal settings used by getty
group	group file
inittab	script for the init file
inode	format of an i-node
issue	issue identification file

ldfcn	common object file access routines	
linenum	line number entries in a common object file	
master	master device information table	
mnttab	mounted file system table	
passwd	password file	
profile	setting up an environment at login time	
reloc	relocation information for a common object file	
sccsfile	format of SCCS file	
scnhdr	section header for a comon object file	
syms	common object file symbol table format	
term	format of compiled term file	
termcap	terminal capability data base	
terminfo	terminal capability data base	
utmp	utmp and wtmp entry formats	
Section 5: Miscellaneous Facilities	5-1
intro	introduction to miscellany	
environ	user environment	
fcntl	file control options	
math	math functions and constants	
modemcap	smart modem capability data base	
piif	performance improvement in large files and direct I/O	
prof	profile within a function	
regex	regular expression compile and match routines	
stat	data returned by stat system call	
term	conventional names for terminals	
types	primitive system data types	
values	machine-dependent values	
varargs	handle variable argument list	

Section 6: Device Files	6-1
intro	introduction to device files
console	console terminal
dsk	winchester, cartridge, and floppy disks
fp	winchester, cartridge, and floppy disks
lp	parallel printer interface
mem	core memory
mt	interface for magnetic tape
null	the null file
prf	operating system profiler
termio	general terminal interface
tp	controlling terminal's local RS-232 channels
tty	controlling terminal interface
window	window management primitives

Tables

1-1	ex Command Names and Abbreviations	1-171
1-2	Determination of SCCS Identification String	1-207
1-3	Identification Keywords and Their Values	1-209
1-4	SCCS Files Data Keywords	1-373
1-5	Octal Codes and Statuses	1-522
3-1	Library Functions	3-4
3-2	Curses Routines	3-39
3-3	Terminfo Level Routines	3-43
3-4	Termcap Compatibility Routines	3-44
3-5	Video Attributes	3-44
3-6	Curses Function Keys	3-45
3-7	Default Error Handling Procedures	3-136
3-8	BTOS File Status Codes	3-160
4-1	Standard Terminal Capabilities	4-65
4-2	Terminal Name Suffixes	4-76
4-3	Capnames and I.codes	4-77
5-1	Errors and Meanings	5-17
5-2	Terminal Names	5-24
6-1	Naming Conventions for Built-In Disk Drives	6-3
6-2	Naming Conventions for SMD Disk Drives	6-4
6-3	Naming Conventions for Tape Drives	6-6

Special File Formats

intro

Name

`intro` - introduction to special file formats

Description

This section outlines the formats of various files. The C struct declarations for the file formats are given where applicable. Usually, these structures can be found in the directories `/usr/include` or `/usr/include/sys`.

a.out

Name

a.out - common assembler and link editor output

Description

The file name a.out is the output file from the assembler **as** and the link editor **ld**. Both programs will make a.out executable if there were no errors in assembling or linking and no unresolved external references.

A common object file consists of a file header, an operating system header, a table of section headers, relocation information, (optional) line numbers, a symbol table, and a string table. The order is given below.

```
File header.  
Operating 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 (line numbers, symbol table and string table) may be missing if the program was linked with the **-s** option of **ld** or if they were removed by **strip**. Also note that if there were no unresolved external references after linking, the relocation information will be absent. The string table exists only if the symbol table contains symbols with names longer than eight characters..

a.out

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

When an a.out file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. The text segment begins at location 0x0000 in the core image. The header is never loaded, except for magic 0413 files created with the **-F** option of **ld**. If the magic number (the first field in the operating system header) is 407 (octal), it indicates that the text segment is not to be write-protected or shared, so the data segment will be contiguous with the text segment. If the magic number is 410 (octal), the data segment and the text segment are not writable by the program; if other processes are executing the same a.out file, the processes will share a single text segment. Magic number 413 (octal) is the same as 410 (octal), except that 413 (octal) permits demand paging. Both the **-z** and **-F** options of the loader **ld** create a.out files with magic numbers 0413. If the **-z** option is used, both the text and data sections of the file are on 1024-byte boundaries. If the **-F** option is used, the text and data sections of the file are contiguous. Loading a single 4096-byte page into memory requires 4 transfers of 1024 bytes each for **-z**, and typically one transfer of 4096 bytes for **-F**. Thus, a.out files created with **-F** can load faster and require less disk space.

The stack begins at the end of memory and grows towards lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the **brk** system call.

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, the storage class of the symbol-table entry for that word will be marked as an "external symbol," and the section number will be set to 0. 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.

a.out

File Header

The format of the filehdr header is

```
struct filehdr
{
    unsigned short f_magic;      /*magic number*/
    unsigned short f_nscns;     /*number of sections*/
    long          f_tmdat;      /*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*/
};
```

Operating System Header

The format of the operating system header is

```
typedef struct southdr
{
    short  magic;      /*magic number*/
    short  vstamp;     /*version stamp*/
    long   tsz;        /*text size in bytes, padded*/
    long   dsz;        /*initialized data (.data)*/
    long   bsz;        /*uninitialized data (.bss)*/
    long   entry;      /*entry point*/
    long   text_start; /*base of text used for file*/
    long   data_start; /*base of data used for file*/
} AOUTHDR;
```

Section Header

The format of the section header is

```
struct schdr
{
    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_innoptr;        /*file ptr to line numbers*/
    unsigned short s_nreloc;        /*# reloc entries*/
    unsigned short s_nlnno;        /*# line number entries*/
    long          s_flags;         /*flags*/
};
```

a.out

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 ref.*/
    long    r_symndx;   /*index into symbol table*/
    short   r_type;     /*relocation type*/
};

```

The start of the relocation information is *s-relptr* from the Section Header. If there is no relocation information, *s-relptr* is 0.

Symbol Table

The format of the symbol table header is

```

#define SYMNMLEN    8
#define FILNMLEN    14
#define SYMESZ      18    /*the size of a SYMENT*/

struct syment
{
    union                /*all ways to get a symbol name*/
    {
        char            _n_name[SYMNMLEN]; /*name of symbol*/
        struct
        {
            long        _n_zeroes; /*==0L if in string table*/
            long        _n_offset; /*location in string table*/
        }_n_n;
        char            *_n_nptr[2]; /*allows overlaying*/
    }_n;
    unsigned long      n_value; /*value of symbol*/
    short              n_scnm; /*section number*/
    unsigned short     n_type; /*type and derived type*/
    char               n_class; /*storage class*/
    char               n_numaux; /*number of aux entries*/
};

#define n_name        _n._n_name
#define n_zeroes      _n._n_n._n_zeroes
#define n_offset      _n._n_n._n_offset
#define n_nptr        _n._n_nptr[1]

```

a.out

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

```

union auxent {
  struct {
    long    x_tegndx;
    union {
      struct {
        unsigned short x_inno;
        unsigned short x_size;
      } x_insz;
      long    x_tsize;
    } x_misc;
    union {
      struct {
        long    x_innoptr;
        long    x_endndx;
      } x_fcn;
      struct {
        unsigned short x_dimen[DIMNUM];
      } x_ary;
    } x_fcary;
    unsigned short x_tvndx;
  } x_sym;

  struct {
    char    x_fname[FILNMLEN];
  } x_file;

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

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

```

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

a.out

See Also

as, **cc**, **ld** in Section 1; **brk** in Section 2; **filehdr**, **ldfcn**, **linenum**, **reloc**, **scnhdr**, **syms** in Section 4.

ar

Name

ar - common archive file format

Description

The archive command **ar** is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor **ld**.

Each archive begins with an archive file header, made up of the following components:

```
#define ARMAG      "<ar>"
#define SARMAG    4

struct ar_hdr {
    char    ar_magic[SARMAG];    /*magic number*/
    char    ar_name[16];        /*archive name*/
    char    ar_date[4];         /*date of last ar. mod.*/
    char    ar_syms[4];         /*no. of ar_sym entries*/
};
```

Each archive that contains common object files (see **a.out**, above) includes an archive symbol table. This symbol table is used by the link editor **ld** to determine which archive members must be loaded during the link edit process. The archive file header described above is followed by a number of symbol table entries. The number of symbol table entries is indicated in the *ar_syms* variable. Each symbol table entry has the following format:

```
struct ar_sym {
    char    sym_name[8];        /*symbol name, recog. by ld*/
    char    sym_ptr[4];        /*archive position of symbol*/
};
```

The archive symbol table is automatically created and/or updated by the **ar** command.

ar

Following the archive header and symbol table are the archive file members. Each file member is preceded by a file member header which is of the following format:

```
struct arf_hdr {           /*archive file member header*/
    char arf_name[16];    /*file member name*/
    char arf_date[4];    /*file member date*/
    char arf_uid[4];     /*file member user ID*/
    char arf_gid[4];     /*file member group ID*/
    char arf_mode[4];    /*file member mode*/
    char arf_size[4];    /*file member size*/
};
```

All information in the archive header, symbol table and file member headers is stored in a machine independent fashion. All character data is automatically portable. The numeric information contained in the headers is also stored in a machine independent fashion. All numeric data is stored as four bytes and is accessed by the special archive I/O functions described under `sputl` in Section 3. Common format archives can be moved from system to system as long as the portable archive command `ar` is used.

Each archive file member begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file, padding excluded.

Notice there is no provision for empty areas in an archive file.

See Also

`ar` and `ld` in Section 1; `sputl` in Section 3.

checklist

Name

checklist - list of file systems processed by **fsck**

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** shell command.

See Also

fsck in Section 1.

core

Name

core - format of core image file

Description

CENTIX writes out a core image of a terminated process when any of various errors occur. See **signal** in Section 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 `/usr/include/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 `/usr/include/sys/user.h`. The important things not detailed therein are the locations of the registers, which are outlined in `/usr/include/sys/reg.h`.

See Also

crash in Section 1; **setuid** and **signal** in Section 2.

cpio

Name

cpio - format of cpio archive

Description

The header structure, when the `-c` option of `cpio` is not used, is:

```
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_filesiz[2];
    char     h_name[h_namesize rounded to word];
} Hdr;
```

When the `-c` option is used, the header information is described by:

```
sscanf(Chdr, "%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%a",
        &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_filesiz*, 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 Section 2, under `stat`. 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_filesiz* equal to zero.

cpio

In PILF files, *h_rdev* contains the cluster size exponent. This should not cause any portability problems, as *h_rdev* is otherwise ignored, except for device special files.

See Also

cpio and **find** in Section 1; **stat** in Section 2; **pilf** in Section 5.

dir

Name

dir - format of directories

Format

```
#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** later in this section). The structure of a directory entry as given in the include file is:

```
#ifndef DIRSIZ
#define DIRSIZ14
#endif
struct    direct
{
        ino_t    d_ino;
        char    d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are `.` 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 `..` and `.` have the same meaning.

See Also

fs in Section 4.

filehdr

Name

filehdr - file header for common object files

Format

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

Description

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

```
struct filehdr
{
    unsigned short    f_magic;      /*magic number*/
    unsigned short    f_nscns;     /*no. of sections*/
    long              f_tmdat;     /*time & date stamp*/
    long              f_symptr;    /*file ptr to symtab*/
    long              f_nsyms;     /* # symtab entries*/
    unsigned short    f_opthdr;    /*sizeof(opt hdr)*/
    unsigned short    f_flags;     /*flags*/
};
```

f_symptr is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in the **fseek** library function to position an I/O stream to the symbol table. The operating system optional header is always 36 bytes. The valid magic numbers are given below. The first three apply to an Application Processor.

```
#define MC68KWRMAGIC 0520 /*writable test segments*/
#define MC68KROMAGIC 0521 /*readonly shareable text segs.*/
#define MC68KPGMAGIC 0522 /*demand paged text segments*/

#define N3BMAGIC 0550 /*3B20S*/
#define NTVMAGIC 0551 /*3B20S*/

#define VAXWRMAGIC 0570 /*VAX writable text segments*/
#define VAXROMAGIC 0575 /*VAX readonlyshareable*/
/*textsegments*/
```

filehdr

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

```
#define F_RELFLG    00001    /*relocation entries stripped*/
#define F_EXEC      00002    /*file is executable*/
#define F_LNNO      00004    /*line numbers stripped*/
#define F_LSYMS     00010    /*local symbols stripped*/
#define F_MINMAL    00020    /*minimal object file*/
#define F_UPDATE    00040    /*update file, ogen produced*/
#define F_SWABD     00100    /*file is "pre-swabbed"*/
#define F_AR16WR    00200    /*16 bit DEC host*/
#define F_AR32WR    00400    /*32 bit DEC host*/
#define F_AR32W     01000    /*non-DEC host*/
#define F_PATCH     02000    /*"patch" list in opt hdr*/
```

See Also

time in Section 2; **fseek** in Section 3; **a.out**.

fs

Name

fs - format of file system

Format

```
#include <sys/filsys.h>
#include <sys/types.h>
#include <sys/param.h>
```

Description

Every file system has a common format for certain vital information. Every such file system 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:

```
/*
 * Structure of the super-block
 */
struct filsys
{
    ushort    s_lsiz; /* size in blocks of l-list */
    daddr_t   s_fsiz; /* size in blocks of file sys */
    short     s_nfree; /* no. of addresses in s_free */
    daddr_t   s_free[NICFREE]; /* free block list */
    short     s_ninode; /* number of i-nodes in s_inode */
    ino_t     s_inode[NICINOD]; /* free i-node list */
    char      s_flock; /* lock during free list manip. */
    char      s_ilock; /* lock during i-list manip. */
    char      s_fmod; /* super block modified flag */
    char      s_ronly; /* mounted readonly flag */
    time_t    s_time; /* last super block update */
    short     s_dinfo[4]; /* device information */
    daddr_t   s_tfree; /* total free blocks */
    ino_t     s_tinode; /* total free i-nodes */
    char      s_fname[6]; /* file system name */
    char      s_fpack[6]; /* file system pack name */
    long      s_fsize[11]; /* ADJUST; make size of filsys */
                /* 512 */
};
```

fs

```

short   s_dummy;    /*reserved for future use*/
short   s_cluster;  /*cluster size (PILF only)*/
long    s_bitsize;  /*size of free block bit map*/
long    s_magic;    /*magic no. to indicate 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 FsPILF       0x10000      /*PILF file system*/

```

CENTIX recognizes three kinds of file systems, specified by *s_type*:

- Oriented to 512-byte I/O. Identified by an *s_type* equal to Fs1b. This type is also assumed if *s_magic* is not equal to FsMAGIC. (This type was originally the only type supported by UNIX Systems; CENTIX does not currently support this type.)
- Oriented to 1024-byte I/O. Identified by an *s_type* equal to Fs2b. This is essentially the standard file system for CENTIX and UNIX System V.
- PILF (Performance Improvement In Large Files) file system. Identified by an *s_type* equal to FsPILF. A PILF file system can be used like a standard file system, but is substantially more efficient when used with direct cluster I/O (see **pilf** in Section 5).

In the following description, the size of a logical block is determined by the file system type. For the original 512-byte oriented file system, a block is 512 bytes. For the 1024-byte oriented file system and the PILF 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.

fs

S_size 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_size* - 2 blocks long. *S_size* 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 presumable corrupted free list.

The free list is provided on 512-byte and 1024-byte file systems, but not on PILF file systems. It 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 an *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.

fs

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_roonly is a read-only 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.

On a PILF file system, *s_cluster* is the default cluster size exponent, used by a process that creates a file on the file system without specifying a default cluster size (see *syslocal* in Section).

I-numbers begin at 1, and the storage for i-nodes begins in block 2. 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* (later in this section).

On a PILF file system, the bit map serves the function of the free list by showing which blocks are allocated to files. It is at the very end of the file system. *S_bitsize* is the number of blocks in the bit map. Each bit in the bit map is 0 if the corresponding 1K data block is allocated to a file.

fs

Files

`/usr/include/sys/filsys.h`
`/usr/include/sys/stat.h`

See Also

fsck, **fsdb**, **mkfs** in Section 1; **inode**; **piif** in Section 5.

fspec

Name

fspec - format specification in text files

Description

It is sometimes convenient to maintain text files on CENTIX with non-standard tabs, (that is, tabs that are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by CENTIX 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:

<i>t</i> tabs	The t parameter specifies the tab settings for the file. The value of <i>tabs</i> 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 <i>n</i> , indicating tabs at intervals of <i>n</i> columns; or 3) A - followed by the name of a "canned" tab specification. Standard tabs are specified by t-8 , or equivalently, t1, 9, 17, 25 , and so on. The canned tabs that are recognized are defined by the tabs shell command (see tabs , Section 1).
<i>s</i> size	The s parameter specifies a maximum line size. The value of <i>size</i> must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.
<i>m</i> margin	The m parameter specifies a number of spaces to be prepended to each line. The value of <i>margin</i> must be an integer.

fspec

- d** The **d** parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.
- e** The **e** parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

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

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

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

Several CENTIX commands correctly interpret the format specification for a file.

See Also

ed, **newform**, **tabs** in Section 1.

gettydefs

Name

gettydefs - speed and terminal settings used by getty

Description

The `/etc/gettydefs` file contains information used by the `getty` shell command 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`, and so on, 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` system call 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` in Section 6). 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`.

gettydefs

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** cannot find the specified label. If `/etc/gettydefs` itself is missing, there is one entry built into the command that will bring up a terminal at 9600 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

getty, **login** in Sec. 1; **ioctl** in Sec. 2; **termio** in Sec. 6.

group

Name

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 IDs to names.

Files

/etc/group

See Also

newgrp and **passwd** in Section 1; **crypt** in Section 3; **passwd** in Section 4.

inittab

Name

inittab - script for the `init` process

Description

The `inittab` file supplies the script to `init`'s role as a general process dispatcher. A separate `inittab` is required for each processor; the last two characters of the name are the processor number. The process that constitutes the majority of `init`'s process dispatching activities is the line process `/etc/getty` that initiates individual terminal lines. Other processes typically dispatched by `init` are daemons and the shell.

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

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

Each entry is delimited by a new-line, however, a backslash (\) preceding a new-line 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` command convention for comments. Comments for lines that spawn `gettys` are displayed by the `who` 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 to four characters used to uniquely identify an entry.

inittab

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 that 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 that have these characters in the *rstate* field are processed only when the *telinit* (see *init* in Section 1) 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, start the process; do not wait for its termination (continue scanning the *inittab* file). When it dies, restart the process. If the process currently exists, do nothing and continue scanning the *inittab* file.

inittab

wait	Upon <code>init</code> 's entering the run-level that matches the entry's <i>rstate</i> , start the process and wait for its termination. All subsequent reads of the <code>inittab</code> file while <code>init</code> is in the same run-level will cause <code>init</code> to ignore this entry.
once	Upon <code>init</code> 's entering a run-level that matches the entry's <i>rstate</i> , 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 <code>init</code> 's boot-time read of the <code>inittab</code> file. <code>init</code> 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 <i>rstate</i> should be the default or it must match <code>init</code> '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 only at <code>init</code> 's boot-time read of the <code>inittab</code> file. <code>init</code> is to start the process, wait for its termination and, when it dies, not restart the process.
powerfail	Execute the process associated with this entry only when <code>init</code> receives a power fail signal (SIGPWR). See <code>signal</code> , Section 2.

inittab

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 <i>rstate</i> field.
initdefault	An entry with this action is only scanned when init is initially invoked. init uses this entry, if it exists, to determine which run-level to enter initially. It does this by taking the highest run-level specified in the <i>rstate</i> field and using that as its initial state. If the <i>rstate</i> field is empty, this is interpreted as 0123456 , causing init to enter run-level 6. Also, the initdefault entry can use s to specify that init start in the single-user state. Additionally, if init doesn't find an initdefault entry in <i>/etc/inittab</i> , then it will request an initial run-level from the user at reboot time.

inittab

sysinit

Entries of this type are executed before `init` tries to access the console. It is expected that this entry will be only used to initialize devices on which `init` might try to ask the run-level question. These entries are executed and waited for before continuing.

process

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

Files

`/etc/inittab??` (last two characters specify the Application Processor)

See Also

`getty`, `init`, `sh`, `who` in Section 1; `exec`, `open`, `signal` in Section 2.

inode

Name

inode - format of an i-node

Format

```
#include <sys/types.h>
#include <sys/ino.h>
```

Description

An i-node for a plain file or directory in a file system has the following structure defined by `<sys/ino.h>`.

```
/*inode structure as it appears on a disk block.*/
struct dinode
{
    ushort di_mode;      /*mode and type of file*/
    short di_nlink;     /*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[39];   /*disk block addresses*/
    char di_cl;         /*PILF cluster size exponent*/
    time_t di_atime;    /*time last accessed*/
    time_t di_mtime;    /*time last modified*/
    time_t di_ctime;    /*time of last file stat 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** in Section 5.

In a PILF file, addresses are organized as in a standard 1K file system, with identical use of blocks of additional addresses. Data addresses, however, do not point to individual 1K blocks; instead, each points to the first block of a contiguous cluster of blocks, each of which is 2^n 1K blocks long, where n is the value in the `di_cl` field.

inode

Files

`/usr/include/sys/ino.h`

See Also

`stat` in Section 2; `fs`; `piif`, `types` in Section 5.

issue

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 that is read by program **getty** and then written to any terminal spawned or respawned from the lines file.

Files

`/etc/issue`

See Also

login in Section 1.

ldfcn

Name

ldfcn - common object file access routines

Format

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

Description

The common object file access routines are a collection of functions for reading an object file that is in common object file form. 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 ldfile, declared in the header file ldfcn.h. The primary purpose of this structure is to provide uniform access to both simple object files and to object files that are members of an archive file.

The library function **ldopen** 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. They contain the following information:

LDFILE *ldptr; TYPE(ldptr)	The file magic number, used to distinguish between archive members and simple object files.
OPTR(ldptr)	The file pointer returned by <i>fopen</i> and used by the standard input/output functions.

ldfcn

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** and **ldaopen** open a common object file.
- **ldclose** and **ldaclose** close a common object file.

2 Functions that read header or symbol table information.

- **ldahread** reads the archive header of a member of an archive file.
- **ldfhread** reads the file header of a common object file.
- **ldshread** and **ldnshread** read a section header of a common object file.
- **ldtbread** reads a symbol table entry of a common object file.

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** seek to the optional file header of a common object file.
- **ldsseek** and **ldnsseek** seek to a section of a common object file.
- **ldrseek** and **ldnrseek** seek to the relocation information for a section of a common object file.
- **ldlseek** and **ldnlseek** seek to the line number information for a section of a common object file.
- **ldtbseek** seek to the symbol table of a common object file.

4 The function **ldtbindex**, which returns the index of a particular common object file symbol table entry

ldfcn

These functions are described in detail in their respective manual pages in Section 3.

All the functions except **ldopen**, **ldaopen** and **ldtbindx** return either **SUCCESS** or **FAILURE**, both constants defined in **ldfcn.h**. **ldopen** and **ldaopen** both return pointers to an **LDFILE** structure.

Macros

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:

```
LDFILE*ldptr;

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)
```

See the manual entries for the corresponding standard input/output library functions for details on the use of these macros.

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

ldfcn

Caution

The macro FSEEK defined in the header file ldfcn.h translates into a call to the standard input/output function fseek. 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!

See Also

fseek, ldahread, ldclose, ldfhread, ldhread, ldseek, ldohseek, ldopen, ldrseek, ldseek, ldshread, ldtbindex, ldtbread, ldtbseek in Section 3.

linenum

Name

linenum - line number entries in a common object file

Format

```
#include <linenum.h>
```

Description

Compilers based on **pcc** generate 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** in Section 1). Users can then reference line numbers when using the appropriate software test system. The structure of these line number entries appears below.

```
struct lineno
{
    union
    {
        long      l_symndx;
        long      l_paddr;
    }
    unsigned short l_inno;
};
```

linenum

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 following:

<i>L_addr</i>	<i>L_Inno</i>
function symtab index	0
physical address	line
physical address	line
....	
function symtab index	0
physical address	line
physical address	line
....	

See Also

cc in Section 1; **a.out**.

master

Name

master - master device information table

Description

This file is used by the **config** program to obtain device information that enables it to generate the configuration files. Do not modify it unless you fully understand its construction. The file consists of 3 parts, each separated by a line with a dollar sign (\$) in column 1. Part 1 contains device information; part 2 contains names of devices that have aliases; part 3 contains tunable parameter information. Any line with an asterisk (*) in column 1 is treated as a comment.

Part 1 contains lines consisting of 6 or 7 fields, with the fields delimited by tabs and/or blanks:

Field 1:	device name (S chars. maximum).
Field 2:	device mask (octal)-each "on" bit indicates that the handler exists: 000100 initialization handler 000040 power-failure handler 000020 open handler 000010 close handler 000004 read handler 000002 write handler 000001 ioctl handler
Field 3:	device type indicator (octal): 000200 allow only one of these devices 000100 suppress count field in conf.c file 000040 suppress interrupt vector 000020 required device 000010 block device 000004 character device 000002 floating vector 000001 fixed vector
Field 4:	handler prefix (4 chars. maximum).
Field 5:	major device number for block-type device
Field 6:	major device number for character-type device
Field 7:	(optional) maximum serial devices on system

master

Part 2 contains lines with 2 fields each:

Field 1: alias name of device (8 chars. maximum).
Field 2: reference name of device (8 chars. maximum; specified in
 part 1).

Part 3 contains lines with 2 or 3 fields each:

Field 1: parameter name (as it appears in description file; 20 chars.
 maximum)
Field 2: parameter name (as it appears in the **conf.c** file; 20 chars.
 maximum)
Field 3: default parameter value (20 chars. maximum; parameter
 specification is required if this field is omitted)

See Also

config in Section 1.

mnttab

Name

mnttab - mounted file system table

Format

```
#include <mnttab.h>
```

Description

Mnttab resides in directory /etc and contains a table of devices, mounted by the **mount** shell command, in the following structure as defined by <mnttab.h>:

```
struct    mnttab {
          char          mt_dev[32];
          char          mt_fsys[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 NMOUNT located in /usr/src/uts/cf/conf.c, which defines the number of allowable mounted special files.

See Also

mount and **setmnt** in Section 1.

passwd

Name

passwd - password file

Description

Passwd contains for each user the following information:

- login name
- encrypted password
- numerical user ID
- numerical group ID
- a field with no standard use
- 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 fifth field exists for historical reasons; it is often used to hold the user's name and address. 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 or her 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.)

passwd

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 or her password has expired will be forced to supply a new one. The next character, *m* say, denotes the minimum period in weeks that must expire before the password may be changed. The remaining characters define the week (counted from the beginning of 1970) when the password was last changed. (A null string is equivalent to zero.) *M* and *m* have numerical values in the range 0-63 that correspond to the 64-character alphabet shown above (that is, / = 1 week; z = 63 weeks). If $m = M = 0$ (derived from the string . or ..) the user will be forced to change his password the next time he or she logs in (and the "age" will disappear from the entry in the password file). If $m > M$ (signified, for example, by the string ./) only the super-user will be able to change the password.

Files

/etc/passwd

See Also

login, passwd in Section 1; a64l, crypt, getpwent in Section 3; group.

profile

Name

profile - setting up an environment at login time

Description

If your login directory contains a file named `.profile`, that file will be executed (via the shell's `exec .profile`) before your session begins; `.profiles` are handy for setting exported environment variables and terminal modes. If the file `/etc/profile` exists, it will be executed for every user before the `.profile`. The following example is typical (except for the comments):

```
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 22
# Tell me when new mail comes in
MAIL = /usr/mail/myname
# Add my /bin directory to the shell search sequence
PATH = $PATH:$HOME/bin
# Set terminal type
export TERM
while true
do
    echo 'terminal: /c'
    read TERM
    if tset
    then
        break
    fi
done
```

Files

`$HOME/.profile`
`/etc/profile`

See Also

`tset`, `env`, `login`, `mail`, `sh`, `stty`, `su` in Section 1; `environ`, `term` in Section 5.

reloc

Name

reloc - relocation information for a common object file

Format

```
#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:

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

/*
 * All generics
 *   reloc. already performed to symbol in the same section
 */
#define R_ABS          0

/*
 * 3B generic
 *   24-bit direct reference
 *   24-bit "relative" reference
 *   16-bit optimized "indirect" TV reference
 *   24-bit "indirect" TV reference
 *   32-bit "indirect" TV reference
 */
#define R_DIR24       04
#define R_REL24       05
#define R_OPT16       014
#define R_IND24       015
#define R_IND32       016

/*
 * DEC Processors VAX 11/780 and VAX 11/750
 * Also Motorola Processors 68000, 68010, and 68020
 */
/*
 */
#define R_RELBYTE     017
#define R_RELWORD     020
#define R_RELLONG     021
#define R_PCRBYTE     022
#define R_PCRWORD     023
#define R_PCRLONG     024
```

reloc

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_DIR24	A direct, 24-bit reference to a symbol's virtual address.
R_REL24	A "PC-relative," 24-bit reference to a symbol's virtual address. Relative references occur in instructions such as jumps and calls. The actual address used is obtained by adding a constant to the value of the program counter at the time the instruction is executed.
R_OPT16	An optimized, indirect, 16-bit reference through a transfer vector. The instruction contains the offset into the transfer vector table to the transfer vector where the actual address of the referenced word is stored.
R_IND24	An indirect, 24-bit reference through a transfer vector. The instruction contains the virtual address of the transfer vector, where the actual address of the referenced word is stored.
R_IND32	An indirect, 32-bit reference through a transfer vector. The instruction contains the virtual address of the transfer vector, where the actual address of the referenced word is stored.
R_RELBYTE	A direct 8-bit reference to a symbol's virtual address.
R_RELWORD	A direct 16-bit reference to a symbol's virtual address.
R_RELLONG	A direct 32-bit reference to a symbol's virtual address.
R_PCRBYTE	A "PC-relative," 8-bit reference to a symbol's virtual address.
R_PCRWORD	A "PC-relative," 16-bit reference to a symbol's virtual address.
R_PCRLONG	A "PC-relative," 32-bit reference to a symbol's virtual address.

On the VAX processors, relocation of a symbol index of -1 indicates that the relative difference between the current segment's start address and the program's load address is added to the relocatable address.

Other relocation types will be defined as they are needed.

reloc

Relocation entries are generated automatically by the assembler and automatically utilized by the link editor. A link editor option exists for removing the relocation entries from an object file.

See Also

ld, strip in Section 1; **a.out, syms**.

sccsfile

Name

sccsfile - format of SCCS file

Description

An SCCS file is an ASCII file. It consists of six logical parts: the *checksum*, the *delta table* (contains information about each delta), *user names* (contains login names and/or numerical group IDs of users who may add deltas), *flags* (contains definitions of internal keywords), *comments* (contains arbitrary descriptive information about the file), and the *body* (contains the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines that 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 DDDDD represent a five-digit string (a number between 00000 and 99999).

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

Checksum

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

@hDDDDD

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

sccsfile

Delta table

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

```

@s DDDDD/DDDDD/DDDDD
@d <type><SCCS ID> yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD

@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
.
.
.
@c <comments> ...
.
.
.
@e

```

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

The **@i**, **@x**, **@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.

sccsfile

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** in Section 1 for more information on their use). Each flag line takes the form:

@f<flag> <optional text>

The following flags are defined:

@ft	<type of program>
@fv	<program name>
@fi	<keyword string>
@fb	
@fm	<module name>
@ff	<floor>
@fc	<ceiling>
@fd	<default-sid>
@fn	
@fj	
@fl	<lock-releases>
@fq	<user defined>
@fz	<reserved for use in interfaces>

sccsfile

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 (for example, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty. The **j** flag causes **get** to allow concurrent edits of the same base SID. The **l** flag defines a list of releases that are locked against editing (**get** with the **-e** keyletter). The **q** flag defines the replacement for the **%Q%** identification keyword. The **s** 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.

sccsfile

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, delta, get, prs in Section 1.

scnhdr

Name

scnhdr - section header for a common object file

Format

```
#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 relocat.*/
    long          s_innoptr; /*file ptr to line #s*/
    unsigned short s_nreloc; /*# reloc entries*/
    unsigned short s_nlnno; /*# line no. 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 Section 3). 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_innoptr*, *s_nreloc*, and *s_nlnno* are zero.

See Also

ld in Section 1; **fseek** in Section 3; **a.out**.

syms

Name

syms - common object file symbol table format

Format

```
#include <syms.h>
```

Description

Common object files contain information to support symbolic software testing (see **sdb** in Section 1). Line number entries (see **linenum**), 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.

syms

The C structure is given below.

```
#define SYMNLEN 8
#define FILNLEN 14
#define DIMNUM 4

struct syment
{
    union /*all ways to get symbol name*/
    {
        char _n_name[SYMNLEN]; /*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;
    long n_value; /*value of symbol*/
    short n_scnum; /*section number*/
    unsigned short n_type; /*type and derived type*/
    char n_class; /*storage class*/
    char n_numaux; /*number of aux entries*/
};

#define n_name _n._n_name
#define n_zeroes _n._n_n._n_zeroes
#define n_offset _n._n_n._n_offset
#define n_nptr _n._n_nptr[1]
```

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

syms

```

union auxent
{
    struct
    {
        long      x_tagndx;
        union
        {
            struct
            {
                unsigned short  x_inno;
                unsigned short  x_size;
            } x_insz;
            long x_fsize;
        } x_misc;
        union
        {
            struct
            {
                long x_innoptr;
                long x_endndx;
            } x_fcn;
            struct
            {
                unsigned short  x_dimen[DIMNUM];
            } x_ary;
        } x_fcary;
        unsigned short x_tvndx;
    } x_sym;
    struct
    {
        char x_fname[FILNMLEN];
    } x_file;
    struct
    {
        long x_scnlen;
        unsigned short x_nreloc;
        unsigned short x_nlinno;
    } x_scn;

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

```

Indexes of symbol table entries begin at zero.

syms

Cautions

CENTIX C longs are equivalent to ints and are converted to ints in the compiler to minimize the complexity of the compiler code generator. Thus, the information about which symbols are declared as longs and which symbols are declared as ints does not show up in the symbol table.

See Also

sdb in Section 1; **a.out**, **linenum**.

term

Name

term - format of compiled term file.

Format

term

Description

Compiled terminfo descriptions are placed under the directory `/usr/lib/terminfo`. In order to avoid a linear search of a huge directory, a two-level scheme is used:

`/usr/lib/terminfo/c/name` where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, `act4` can be found in the file `/usr/lib/terminfo/a/act4`. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it will be the same on all hardware. An 8 or more bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the `compile` program, and read by the routine `setupterm`. Both of these pieces of software are part of `curses` (see Section 3). 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.

term

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 $256 * \text{second} + \text{first}$.) The value -1 is represented by 0377, 0377; other negative values are illegal. The -1 generally means that a capability is missing from this terminal. Note that this format corresponds to the hardware of the VAX and PDP-11. Machines where this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the "|" character. The section is terminated with an ASCII NUL character.

The Boolean flags have one byte for each flag. This byte is either 0 or 1 as the flag is present or absent. 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 flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is -1, 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 means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in $\backslash X$ or $\backslash c$ 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.

term

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 routing **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 Microterm ACT 4 is included:

```
microterm|act4|microterm act iv,
cr=^M, cud1=^J, ind=^J, bel=^G, am, cub1=^H,
ed=^_, el=^^, clear=^L, cup=^T%p1%c%p2%c,
cols#60, lines#24, cuf1=^X, cuu1=^Z, home=^],

000 032 001  \0 025 \0 \b \0 212 \0 " \0 m i c r
020 o t e r m | a c t 4 | m i c r o
040 t e r m a c t i v \0 \0 001 \0 \0
080 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
100 \0 \0 P \0 377 377 030 \0 377 377 377 377 377 377 377
120 377 377 377 377 \0 \0 002 \0 377 377 377 377 004 \0 006 \0
140 \b \0 377 377 377 377 \n \0 026 \0 030 \0 377 377 032 \0
160 377 377 377 377 034 \0 377 377 036 \0 377 377 377 377 377
200 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
520 377 377 377 377 \0 377 377 377 377 377 377 377 377 377
540 377 377 377 377 377 007 \0 \r \0 \f \0 036 \0 037 \0
560 024 % p 1 % c % p 2 % c \0 \n \0 035 \0
600 \b \0 030 \0 032 \0 \n \0
```

Some limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

Files

`/usr/lib/terminfo/*/*` - compiled terminal capability data base

See Also

curses in Section 3; **terminfo**.

termcap

Name

termcap - terminal capability data base

Format

`/etc/termcap`

Description

This entry describes terminal-independent programming conventions that originate at UC Berkeley. UNIX System V initially borrowed termcap but has since changed to the terminfo convention. CENTIX continues to support termcap so as to be compatible with the Berkeley version of the UNIX System, but use terminfo in new programs.

Termcap programs work from information supplied through the TERM and TERMCAP environment variables. The location of the description depends on the value of TERMCAP.

- If TERMCAP is not set or is empty, TERM is the name of a description in `/etc/termcap`.
- If TERMCAP has a value that begins with a `/`, TERM is the name of a description in the file named by TERMCAP.
- If TERMCAP begins with any character except `/`, TERMCAP contains the description.

A description begins with a list of its names, separated by vertical bars. The rest of the description is a list of capabilities, separated by colons. If you use more than one line, precede each new-line except the last with `:\`. Here's a simple example.

```
d5 vt50 dec vt5:\
:bs:cd=\EJ:ce=\EK:cl=\EH\EJ:co#80:li#12:nd=\EC:pt:up=/EA:
```

termcap

There are three kinds of capabilities:

- Boolean. These indicate the presence or absence of a terminal feature by their presence or absence. Boolean capabilities consist of two characters (the capability name).
- Numeric. These indicate some numeric value for the terminal, such as screen size or delay required by a standard character. Numeric capabilities consist of two characters (the capability name), followed by a #, followed by a decimal number.
- String. These indicate a sequence that performs some operation on the terminal. String capabilities consist of two characters (the capability name), optionally followed by a delay, followed by a string.

The delay is the number of milliseconds the program must wait after using the sequence; specify no more than one decimal place. If the delay is proportional to the number of lines affected, end it with a *.

The string is a sequence of characters. The following subsequences are specially interpreted.

<code>\E</code>	Escape Character
<code>^x</code>	Control-x
<code>\n</code>	Newline
<code>\r</code>	Return
<code>\t</code>	Tab
<code>\b</code>	Backspace
<code>\f</code>	Formfeed
<code>\xxx</code>	Octal value of xxx
<code>\072</code>	in string
<code>\200</code>	null (<code>\000</code> doesn't work)

Octal numbers must be three digits long.

Some strings are interpreted further, such as `cm`. See below.

termcap

You can follow any capability name with @ to indicate that the terminal lacks the capability. This is only useful in conjunction with the `tc` capability; see "Similar Terminals," below.

Table 4-1 is a list of standard capabilities. (P) indicates a string that might require padding; (P*) indicates a string that might require proportional padding.

Table 4-1 Standard Terminal Capabilities

Name	Type	Pad?	Description
ae	str	(P)	Ends alternate character set.
al	str	(P*)	Adds new blank line.
am	bool		Terminal has automatic margins.
as	str	(P)	Starts alternate character set.
bc	str		Backspace if not control-h.
bs	bool		Terminal can backspace with control-h.
bt	str	(P)	Back tab.
bw	bool		Backspace wraps from column 0 to last column.
CC	str		Command character in prototype if terminal is settable.
cd	str	(P*)	Clears to end of display.
ce	str	(P)	Clears to end of line.
ch	str	(P)	Moves cursor horizontally to specified column.
cl	str	(P*)	Clears screen.
cm	str	(P)	Moves cursor to specified row and column.
co	num		Number of columns in a line.
cr	str	(P*)	Carriage return if not control-m.
cs	str	(P)	Change scrolling region.
cv	str	(P)	Moves cursor vertically to a specified row.
da	bool		Display can be retained above.
dB	num		Delay after backspace, in milliseconds.
db	bool		Display can be retained below.
dC	num		Delay after carriage return, in milliseconds.
dc	str	(P*)	Delete character.
dF	num		Delay after form feed, in milliseconds.
dl	str	(P*)	Deletes line.
dm	str		Enters delete mode.
dN	num		Delay after new-line, in milliseconds.

termcap

Table 4-1 Standard Terminal Capabilities (Cont.)

Name	Type	Pad?	Description
do	str		Goes down one line.
dT	num		Delay after tab, in milliseconds.
ed	str		Ends delete mode.
ei	str		Ends insert mode; give an empty string if you've defined ic.
eo	str		Can erase overstrikes with a blank.
ff	str	(P*)	Hardcopy terminal page eject if not form feed.
hc	bool		Hardcopy terminal.
hd	str		Half-line down (forward 1/2 linefeed).
ho	str		Move cursor to upper left corner (home).
hu	str		Half-line up (reverse 1/2 linefeed).
hz	str		Hazeltine or other terminal that can't print ~s.
ic	str	(P)	Insert character.
if	str		Name of file containing terminal initialization.
im	bool		Starts insert mode; give an empty string if you've defined ic.
in	bool		Insert mode distinguishes nulls on display.
ip	str	(P*)	Pad after insertion.
is	str		Terminal initialization.
k0-k9	str		Sent by special (usually numeric) function keys. If programmable, set with is, if, vs, or ti.
kb	str		Sent by backspace key.
kd	str		Sent by terminal down arrow key.
ke	str		Ends keypad transmit code.
kh	str		Sent by home key.
kl	str		Sent by terminal left arrow key.
kn	num		Number of special function keys.
ko	str		Terminal capabilities that have keys.
kr	str		Sent by terminal right arrow key.
ks	str		Begin keypad transmit mode.
ku	str		Sent by terminal up arrow key.
l0-l9	str		Labels on special function keys.
li	str		Last line, first column.
ma	str		Command key map; used by ex version 2.
mi	bool		Safe to move while in insert mode.
ml	str		Memory lock on above cursor.
ms	bool		Safe to move while in standout or underline mode.
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correctly working carriage return.
nd	str		Non-destructive space (cursor right).
nl	str	(P*)	Begin a new line if not new-line.
ns	bool		A video terminal that doesn't scroll.
os	bool		Terminal overstrikes.

termcap

Table 4-1 Standard Terminal Capabilities (Cont.)

Name	Type	Pad?	Description
pc	str		Pad character if not null.
pt	bool		Has hardware tabs; if they need to be set, put sequence in is or if.
se	str		Ends stand out mode.
sf	str	(P)	Scrolls forward.
sg	num		Number of blank characters left by so or se.
so	str		Begins stand out mode.
sr	str	(P)	Scroll reverse (backwards).
ta	str	(P)	Tab if not control-i or with padding.
tc	str		Name of terminal that has some of the same capabilities; tc must be the last capability.
te	str		Ends programs that do cursor motion.
ti	str		Initializes programs that do cursor motion.
uc	str		Underscores and moves past one character.
ue	str		Ends underscore mode.
ug	num		Number of blank spaces that surround underscore mode.
ul	bool		Terminal underlines automatically even though it can't overstrike.
up	str		Upline (cursor up).
us	str		Start underscore mode.
vb	str		Visible bell (must not move cursor).
ve	str		Ends open and visual modes.
vs	str		Initializes open and visual modes.
xb	bool		Beehive (f1=escape, f2=ctrl C).
xn	bool		Terminal ignoresc new-line after wrap (Concept).
xr	bool		Returns clears to end of line and goes to beginning of next line (Delta Data).
xs	bool		Writing on standout mode text produces standout mode text (HP 264?)
xt	bool		Destructive tabs, magic standout character (Teleray 1061).

Pointers on Preparing Descriptions

- You may want to copy the description of a similar terminal.
- Build up a description gradually, checking partial descriptions with `ex`.
- Be aware that an unusual terminal may expose bugs in `ex` limitations in the termcap convention.

termcap

Basic Capabilities

The following capabilities are common to most terminals. The **co** capability gives the number of columns per line. The **li** gives the number of lines on a video terminal. The **am** capability indicates that writing off the right edge takes the cursor to the beginning of the next screen. The **cl** capability tells how the terminal clears its screen. The **bs** indicates that the terminal can backspace; but if the terminal doesn't use control-h, specify **bc** instead of **bs**. The **os** capability indicates that printing a character at an occupied position doesn't destroy the existing character.

A couple of notes on moving off the edge. Programs that use this convention never move the cursor off the top or the left edge of the screen. On the other hand, they assume that moving off the bottom edge scrolls the display up.

These capabilities suffice to describe hardcopy and very dumb terminals.

Cursor Addresses and Other Variables

If a string capability includes a variable value, use a % escape to indicate the value. By default, programs take these values to be zero origin (that is, the first possible value is 0) and that the **cm** capability specifies two values: row, then column. Use the **%r** or **%i** capability if either assumption is incorrect.

These are the valid % escapes.

%d	Print the values as a decimal number.
%2	Print the values as a two-digit decimal number.
%3	Print the values as a three-digit decimal number.
%.	Print the value in binary (but see below).
%+x	Add ASCII value of x to value, then print in binary.
%>xy	If the next value is greater than the ASCII value of x, add the ASCII value of y before using the value's % escape.

termcap

%r	Row is the first value in this cm .
%i	Values are 1-origin.
%%	Print a %.
%n	In this capability, exclusive or the values with 01400 before using the values' % escapes (DM2500).
%B	Change the next value to binary coded decimal ($(16^*(x/10) + (x\%10))$ where x is the value) before interpreting it.
%D	The next value is reverse-coded ($x-2^*(x\%16)$ where x is the value; Delta Data)

A program should avoid using a **cm** sequence that includes a tab, new-line, control-d, or return, because the terminal interface may misinterpret these characters. If possible, use the **cm** sequence to move to the row or column after the destination, then use local motion to get to the destination.

Here are some examples of **cm** definitions. To position the cursor of an HP2645 on row 3, column 12, you must send the terminal "\E&a12c03Y," followed by a 6 millisecond delay; the HP2645 description includes :cm=6\E&%r%2c%2Y:. To position the cursor of an ACT-IV, you send it a control-t, followed by the row and column in binary; the ACT-IV description includes :cm=^T%.%.:. The LSI ADM3a uses the set of printable ASCII characters to represent row and column values; its description includes :cm\x=%+%+:.:

Local and General Cursor Motions

Most terminals have short strings that trigger commonly-used cursor motions. A non-destructive space (BR nd) moves the cursor one position right. An upline sequence (up) moves the cursor one position up. A home sequence (ho) moves the cursor to the upper left hand corner. A lower-left (ll) goes to the other left hand corner. The ll capability may be a sequence that moves the cursor home, then up; but otherwise programs never do this.

termcap

Area Clears

Some terminals have short sequences that clear all or part of a display. Clear (**cl**) clears the screen and homes the cursor; if clearing the screen does not restore the terminal's normal modes, **cl** should include the strings that do. Clear to end of line (**ce**) clears from the current cursor position to the right. Clear to end of display (**cd**) clears from the current cursor position to the bottom of the display; programs always move the cursor to the beginning of the line before using **cd**.

Insert/Delete Line

Many terminals have strings that shift text starting at the current cursor position. Programs always move the cursor to the beginning of the line before using these strings. Add line (**al**) shifts the current line and all below it down a position leaving the cursor on the newly-blanked line. Delete line (**dl**) deletes the line the cursor is on without moving the cursor. If a terminal description has an **al** capability, you do not really need to specify **sb**.

If deleting a line might produce a non-blank line at the bottom of the screen, specify **db**. If scrolling backwards might produce a non-blank line at the top of the screen, specify **da**.

Insert/Delete Character

The termcap convention recognizes two kinds of terminal insert/delete string.

- The first convention is by far more common. Using insert or delete modes only affect characters on the current line. Inserting a single character shifts all characters, including all blanks, to the right; the character on the right edge of the screen is lost. No special capability is required to describe this kind of terminal.

termcap

- The second convention is rarer and more complicated. The terminal distinguishes between blank spaces created by output tabs (O11) or spaces (O40) from all other blanks; other blanks are known as nulls. Inserting a character eliminates the first null to the right of the cursor; deleting a character doubles the first null. If there are no nulls on the current line, inserting a character inserts the line's rightmost character at the beginning of the next line. Use the **in** capability to describe this kind of terminal.

A simple experiment shows what type you have. Set the terminal to its "local" mode. Clear the screen, then type a short sequence of text. Move the cursor to the right several spaces without using the space or tab characters. Type a second short sequence of text. Move the cursor back to the beginning of the first text. Start the terminal's insert mode and begin tapping the space bar. If you have the first kind of terminal, both sequences of text will move at once; whatever character is at the right edge of the screen will be lost. If you have the second kind of terminal, at first only the first sequence of text will move; when the first sequence hits the second sequence, it will push the second onto the next line.

A terminal can have either an insert mode or the ability to insert a single character. Specify insert mode with **im** and **ei**. To specify that the terminal can insert a single character, specify **ic** and specify empty strings for **im** and **ei**. If you must delay or output more control text after inserting a single character, specify **ip**.

If a terminal has both an insert mode and the ability to insert a single character, it is usually best not to specify **ic**.

Some programs operate more quickly if they are allowed to move the cursor around randomly while in insert mode. For example, **vi** has to delete a character when you insert a character before a tab. If your terminal permits this, specify move on insert **mi**. Beware of terminals that foul up in subtle ways when you do this.

Delete mode (**dm**), end delete mode (**ed**), and delete character (**dc**) work like **im**, **ei** and **ic**.

termcap

Highlighting, Underlining, and Visible Bells

Specify the terminals most distinctive display mode with **so**. Half intensity is usually not a good choice unless the terminal is normally in reverse video.

The convention provides for underline mode and for single character underlining. Specify underline mode with **us** and **ue**. Specify a way to underline and move past a character with **uc**; if your terminal can underline a single character but doesn't automatically move on, add a nondestructive space to the **uc** string.

Some terminals can't overstrike but still correctly underline text without special help from the host computer. If yours is one, specify **ul**.

If your terminal spaces before and after entering standout and underline mode, specify **ug**.

Programs leave standout and underline mode before moving the cursor or printing a new-line.

If the terminal can flash the screen without moving the cursor, specify **vb** (visual bell).

If the terminal needs to change working modes before entering the open and visual modes of **ex** and **vi**, specify **vs** and **ve**, respectively. These can be used to change, for example, from an underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, specify **ti** and **te**. This may be important if a terminal has more than one page of memory. If the terminal has memory-relative cursor addressing but not screen relative cursor addressing, use **ti** to fix a screen-sized window into the terminal.

If a terminal can overstrike, programs assume that printable spaces don't destroy anything, unless you specify **eo**.

termcap

Keypad

Some terminals have keypads that transmit special codes. If the keypad can be turned on and off, specify **ks** and **ke**; if you don't, programs assume that the keypad is always on. Specify the codes sent by cursor motion keys with **kl**, **kr**, **ku**, **kd**, and **kh**. If there are function keys, specify the codes they send with **f1**, **f2**, **f3**, **f4**, **f5**, **f6**, **f7**, **f8**, and **f9**. If these keys have labels other than the usual "f0 through f9," specify the labels **l1**, **l2**, **l3**, **l4**, **l5**, **l6**, **l7**, **l8**, and **l9**. If there are other keys that transmit the same code that the terminal expects for a function, such as clear screen, mention the affected capabilities in the **ko** capability. For example, `":ko=cl,ll,sf,sb:"` says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb capabilities.

Terminal Initialization

If a terminal must be initialized, on login for example, specify a short string with **is** or a file containing initialization strings with **if**. Other capabilities include **is**, and initialization string for the terminal, and **if**, the name of a file containing long initialization strings. If both are given, **is** is printed before **if**. If the terminal has tab stops, these strings should first clear all stops, then set new stops at the 9 column and every 8 column thereafter.

Similar Terminals

If a new terminal strongly resembles an existing terminal, you can write a description of the new terminal that only mentions the old terminal and the capabilities that differ. The **tc** capability describes the old terminal; it must be the last capability in the description. If the old terminal has capabilities that the new one lacks, specify an **@** after the capability name.

termcap

The different entries you create with **tc** need not represent terminals that are actually different. They can represent different uses for a single terminal, or user preferences as to which terminal features are desirable.

The following example defines and describes a variant of the 2621 that never turns on the keypad.

```
hn 2621nl:ks@:ke@:tc=2621:
```

Files

/etc/termcap - standard data base

Known Problems

The **ex** command allows only 256 characters for string capabilities, and the routines in the termcap library function do not check for overflow of this buffer.

The total length of a single description (excluding only escaped new-lines) may not exceed 1024 characters. If you use **tc**, the combined description may not exceed 1024 characters.

The **vs**, and **ve** entries are specific to the **vi** program.

Not all programs support all entries. There are entries that are not supported by any program.

The **ma** capability is obsolete and serves no function in our database; Berkeley includes it for the benefit of systems that cannot run version 3 of **vi**.

See Also

ex, **tset**, **vi**, **ul**, **more** in Section 1; **curses**, **termcap** in Section 3.

terminfo

Name

terminfo - terminal capability data base

Format

```
/usr/lib/terminfo/*/*
```

Description

Terminfo is a data base describing terminals used, for example, by the **vi** command and the **curses** library function. Terminals are described in terminfo by giving a set of capabilities that they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in terminfo.

Entries in terminfo consist of a number of ',' separated fields. White space after ',' is ignored. The first entry for each terminal gives the names that are known for the terminal, separated by '|' characters. The first name given is the most common abbreviation for the terminal, the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should be in lower case and contain no blanks; the last name may well contain upper case and blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the conventions shown in Table 4-2. The particular piece of hardware making up the terminal should have a root name chosen, thus "hp2621." This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. Thus, a vt100 in 132 column mode would be vt100-w. The following suffixes should be used where possible:

terminfo

Table 4-2 Terminal Name Suffixes

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	vt100-w
-am	With auto. margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
-n	Number of lines on the screen	aaa-60
-na	No arrow keys (leave them in local)	c100-na
-np	Number of pages of memory	c100-4p
-rv	Reverse video	c100-rv

Capabilities

The variable is the name by which the programmer (at the terminfo level) accesses the capability. The capname is the short name used in the text of the database, and is used by a person updating the database. The i.code is the two letter internal code used in the compiled database, and always corresponds to the old termcap capability name.

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short and to allow the tabs in the source file caps to line up nicely. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification. For the capnames and i.codes listed in Table 4-3:

- (P) Indicates that padding may be specified.
- (G) Indicates that the string is passed through tparm withparms as given (#i).
- (*) Indicates that padding may be based on the number of lines affected.
- (#i) Indicates the *i*th parameter.

terminfo

Table 4-3 Capnames and l.codes

Variable Booleans	Cap-name	l. code	Description
auto_left_margin,	bw	bw	cut1 wraps from column 0 to last column
auto_right_margin, beehive_glitch,	am xsb	am xb	Terminal has automatic margins Beehive (f1=escape, f2=ctrl C)
ceol_standout_glitch,	xhp	xs	Standout not erased by overwriting (hp)
eat_newline_glitch,	xenl	xn	new-line ignored after 80 cols (Concept)
erase_overstrike,	eo	eo	Can erase overstrikes with a blank
generic_type,	gn	gn	Generic line type (such as dialup, switch)
hard_copy, has_meta_key,	hc km	hc km	Hardcopy terminal Has a meta key (shift, sets parity bit)
has_status_line, insert_null_glitch, memory_above,	hs in da	hs in da	Has extra "status line" Insert mode distinguishes nulls Display may be retained above the screen
memory_below,	db	db	Display may be retained below the screen
move_insert_mode, move_standout_mode,	mir msgr	mi ms	Safe to move while in insert mode Safe to move in standout modes
over_strike, status_line_esc_ok,	os eslok	os es	Terminal overstrikes Escape can be used on the status line
teleray_glitch,	xt	xt	Tabs ruin, magic so char (Teleray 1061)
tilde_glitch, transparent_underline, xon_xoff,	hz ul xon	hz ul xo	Hazeltine, can not print ~s Underline character overstrikes Terminal uses xon/xoff handshaking
Numbers:			
columns,	cols	co	Number of columns in a line
init_tabs,	it	it	Tabs initially every # spaces
lines,	lines	li	Number of lines on screen or page

terminfo

Table 4-3 Capnames and I.codes (Cont.)

Variable Booleans	Cap-name	I. code	Description
lines_of_memory,	lm	lm	Lines of memory if > lines. 0 means varies.
magic_cookie_glitch,	xmc	sg	Number of blank characters left by smso or rmso
padding_baud_rate,	pb	pb	Lowest baud where cr/nl padding is needed
virtual_terminal,	vt	vt	Virtual terminal number (CENTIX system)
width_status_line,	wsl	ws	Number of columns in status line
Strings:			
back_tab,	cbt	bt	Back tab (P)
bell,	bel	bl	Audible signal (bell) (P)
carriage_return,	cr	cr	Carriage return (P*)
change_scroll_region,	csr	cs	change to lines #1 through #2 (vt100) (PG)
clear_all_tabs,	ttc	ct	Clear all tab stops (P)
clear_screen,	clear	cl	Clear screen and home cursor (P*)
clr_eol,	el	ce	Clear to end of line (P)
clr_eos,	ed	cd	Clear to end of display (P*)
column_address,	hpa	ch	Set cursor column (PG)
command_character,	cmdch	CC	Term. settable cmd char in prototype
cursor_address,	cup	cm	Screen rel. cursor motion row #1 col #2 (PG)
cursor_down,	cud1	do	Down one line
cursor_home,	home	ho	Home cursor (if no cup)
cursor_invisible,	civis	vi	Make cursor invisible
cursor_left,	cub1	le	Move cursor left one space
cursor_mem_address,	mrcup	CM	Memory relative cursor addressing
cursor_normal,	cnorm	ve	Make cursor appear normal (undo vs/vi)
cursor_right,	cuf1	nd	Non-destructive space (cursor right)
cursor_to_ll,	ll	ll	Last line, first column (if no cup)
cursor_up,	cuu1	up	Upline (cursor up)
cursor_visible,	cvvis	vs	Make cursor very visible
delete_character,	dch1	dc	Delete character (P*)
delete_line,	dli	dl	Delete line (P*)

terminfo

Table 4-3 Capnames and l.codes (Cont.)

Variable Booleans	Cap-name	l. code	Description
dis_status_line,	dsl	ds	Disable status line
down_half_line,	hd	hd	Half-line down (forward 1/2 linefeed)
enter_alt_charset_mode,	smacs	as	Start alternate character set (P)
enter_blink_mode,	blink	mb	Turn on blinking
enter_bold_mode,	bold	md	Turn on bold (extra bright) mode
enter_ca_mode,	smcup	ti	String to begin programs that use cup
enter_delete_mode,	smdc	dm	Delete mode (enter)
enter_dim_mode,	dim	mh	Turn on half-bright mode
enter_insert_mode,	smir	im	Insert mode (enter)
enter_protected_mode,	prot	mp	Turn on protected mode
enter_reverse_mode,	rev	mr	Turn on reverse video mode
enter_secure_mode,	invis	mk	Turn on blank mode (chars invisible)
enter_standout_mode,	smso	so	Begin stand out mode
enter_underline_mode,	smul	us	Start underscore mode
erase_chars,	ech	ec	Erase # 1 characters (PG)
exit_alt_charset_mode,	rmacs	ae	End alternate character set (P)
exit_attribute_mode,	sgrO	me	Turn off all attributes
exit_ca_mode,	rmcup	te	String to end programs that use cup
exit_delete_mode,	rmdc	ed	End delete mode
exit_insert_mode,	rmir	ei	End insert mode
exit_standout_mode,	rmso	se	End stand out mode
exit_underline_mode,	rmul	ue	End underscore mode
flash_screen,	flash	vb	Visible bell (may not move cursor)
form_feed,	ff	ff	Hardcopy terminal page eject (P*)
from_status_line,	fsl	fs	Return from status line
init_1string,	is1	i1	Terminal initialization string
init_2string,	is2	i2	Terminal initialization string
init_3string,	is3	i3	Terminal initialization string
init_file,	if	if	Name of file containing is
insert_character,	ich1	ic	Insert character (P)
insert_line,	il1	al	Add new blank line (P*)
insert_padding,	ip	ip	Insert pad after character inserted (P*)
key_backspace,	kbs	kb	Sent by backspace key
key_catab,	ktbc	ka	Sent by clear-all-tabs key
key_clear,	kclr	kC	Sent by clear screen or erase key

terminfo

Table 4-3 Capnames and I.codes (Cont.)

Variable Booleans	Cap-name	I. code	Description
key_ctab,	kctab	kt	Sent by clear-tab key
key_dc,	kdch1	kD	Sent by delete character key
key_dl,	kdll	kL	Sent by delete line key
key_down,	kcud1	kd	Sent by terminal down arrow key
key_eic,	krmir	kM	Sent by rmir or smir in insert mode
key_eol,	kel	kE	Sent by clear-to-end-of-line key
key_eos,	ked	kS	Sent by clear-to-end-of-screen key
key_f0,	kf0	k0	Sent by function key f0
key_f1,	kf1	k1	Sent by function key f1
key_f10,	kf10	ka	Sent by function key f10
key_f2,	kf2	k2	Sent by function key f2
key_f3,	kf3	k3	Sent by function key f3
key_f4,	kf4	k4	Sent by function key f4
key_f5,	kf5	k5	Sent by function key f5
key_f6,	kf6	k6	Sent by function key f6
key_f7,	kf7	k7	Sent by function key f7
key_f8,	kf8	k8	Sent by function key f8
key_f9,	kf9	k9	Sent by function key f9
key_home,	khome	kh	Sent by home key
key_ic,	kich1	kl	Sent by ins char/enter ins mode key
key_il,	kil1	kA	Sent by insert line
key_left,	kcub1	k1	Sent by terminal left arrow key
key_ll,	kll	kH	Sent by home-down key
key_npage,	knp	kN	Sent by next-page key
key_ppage,	kpp	kP	Sent by previous page key
key_right,	kcuf1	kr	Sent by terminal right arrow key
key_sf,	kind	kF	Sent by scroll-forward /down key
key_sr,	kri	kR	Sent by scroll-backward /up key
key_stab,	khts	kT	Sent by set-tab key
key_up,	kcuu1	ku	Sent by terminal up arrow key
keypad_local,	rmkx	ke	Out of "keypad transmit" mode
keypad_xmit,	smkx	ks	Put terminal in "keypad transmit" mode
lab_f0,	lf0	l0	Labels on function key f0 if not f0
lab_f1,	lf1	l1	Labels on function key f1 if not f1
lab_f10,	lf10	la	Labels on function key f10 if not f10

terminfo

Table 4-3 Capnames and l.codes (Cont.)

Variable Booleans	Cap-name	l. code	Description
lab_f2,	lf2	l2	Labels on function key f2 if not f2
lab_f3,	lf3	l3	Labels on function key f3 if not f3
lab_f4,	lf4	l4	Labels on function key f4 if not f4
lab_f5,	lf5	l5	Labels on function key f5 if not f5
lab_f6,	lf6	l6	Labels on function key f6 if not f6
lab_f7,	lf7	l7	Labels on function key f7 if not f7
lab_f8,	lf8	l8	Labels on function key f8 if not f8
lab_f9,	lf9	l9	Labels on function key f9 if not f9
meta_on,	smm	mm	Turn on "meta mode" (8th bit)
meta_off,	rmm	mo	Turn off "meta mode"
newline,	nel	nw	New-line (behaves like cr followed by lf)
pad_char,	pad	pc	Pad character (rather than null)
parm_dch,	dch	DC	Delete #1 chars (PG*)
parm_delete_line,	dl	DL	Delete #1 lines (PG*)
parm_down_cursor,	cud	DO	Move cursor down #1 lines (PG*)
parm_ich,	ich	IC	Insert #1 blank chars (PG*)
parm_index,	indn	SF	Scroll forward #1 lines (PG)
parm_insert_line,	il	AL	Add #1 new blank lines (PG*)
parm_left_cursor,	cub	LE	Move cursor left #1 spaces (PG)
parm_right_cursor,	cuf	RI	Move cursor right #1 spaces (PG*)
parm_rindex,	rin	SR	Scroll backward #1 lines (PG)
parm_up_cursor,	cuu	UP	Move cursor up #1 lines (PG*)
pkey_key,	pfkey	pk	Prog funct key #1 to type string #2
pkey_local,	pfloc	pl	Prog funct key #1 to execute string #2
pkey_xmit,	px	px	Prog funct key #1 to xmit string #2
print_screen,	mc0	ps	Print contents of the screen
prtr_off,	mc4	pf	Turn off the printer
prtr_on,	mc5	po	Turn on the printer
repeat_char,	rep	rp	Repeat char #1 #2 times (PG*)
reset_1string,	rs1	r1	Reset terminal completely to sane modes
reset_2string,	rs2	r2	Reset terminal completely to sane modes

terminfo

Table 4-3 Capnames and I.codes (Cont.)

Variable Booleans	Cap-name	I. code	Description
reset_3string,	rs3	r3	Reset terminal completely to sane modes
reset_file,	rf	rf	Name of file containing reset string
restore_cursor,	rc	rc	Restore cursor to position of last sc
row_address,	vpa	cv	Vertical position absolute (set row) (PG)
save_cursor,	sc	sc	Save cursor position (P)
scroll_forward,	ind	sf	Scroll text up (P)
scroll_reverse,	ri	sr	Scroll text down (P)
set_attributes,	sgr	sa	Define the video attributes (PG9)
set_tab,	hts	st	Set a tab in all rows, current column
set_window,	wind	wi	Current window is lines #1-#2 cols #3-#4
tab,	ht	ta	Tab to next 8 space hardware tab stop
to_status_line,	tsl	ts	Go to status line, column # 1
underline_char,	uc	uc	Underscore one char and move past it
up_half_line,	hu	hu	Half-line up (reverse 1/2 linefeed)
init_prog,	iprog	iP	Path name of program for init
key_a1,	ka1	K1	Upper left of keypad
key_a3,	ka3	K3	Upper right of keypad
key_b2,	kb2	K2	Center of keypad
key_c1,	kc1	K4	Lower left of keypad
key_c3,	kc3	K5	Lower right of keypad
prtr_non,	mc5p	p0	Turn on the printer for # 1 bytes

terminfo

A Sample Entry

The following entry, which describes the Concept-100, is among the more complex entries in the terminfo file as of this writing.

```
concept100 | c100 | concept | c104 | c100-4p | concept 100,
am, bel=^G, blank=\EH, blink=\EC, clear=^L$<2^>, cnorm=\Ew,
cols#80, cr=^M$<9>, cub1=^H, cud1=^J, cufl1=\E,
cup=\Ea%p1%'%+%c%p2%'%+%c,
cuu1=\E; cvvis=\EW, db, dch1=\E^A$<16^>,
dlm=\EE, dl1=\E^B$<3^>,
ed=\E^C$<16^>, el=\E^U$<16^>, eo, flash=\Ek$<20>\EK,
ht=\t$<8>,
ll1=\E^R$<3^>, ln, ind=^J, ind=^J$<9>, lp=$<16^>,
ls2=\EU\EI\E7\E5\E8\EI\ENH\EK\E200\Eo&\200\Eo\47\E,
kbs=^h, kcb1=\E>, kcud1=\E<, kcufl1=\E=, kcuu1=\E; ,
kf1=\E5, kf2=\E6, kf3=\E7, khome=\E?
lines#24, mir, pb#9800, prot=\EI, rep=\Er%p1%c%p2%'%+%c$<.2^>,
rev=\ED, rmcup=\Ev $<8>\Ep\r\n, rmlr=\E\200, rmkx=\Ex,
rmso=\Ed\Ee, rmul1=\Eg, rmul2=\Eg, sgr0=\EN\200,
smcup=\EU\Ev 8p\Ep\r, smlr=\E^P, smkx=\EX, smso=\EE\ED,
smul1=\EG, tabs, ul, vt#8, xenl,
```

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Comments may be included on lines beginning with “#.” Capabilities in terminfo are of three types: Boolean capabilities, which indicate that the terminal has some particular feature; numeric capabilities giving the size of the terminal or the size of particular delays; and string capabilities, which give a sequence that 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 (that is, 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.

terminfo

Finally, string valued capabilities, such as `el` (clear to end of line sequence) are given by the two-character code, an '=', and then a string ending at the next following ';'. A delay in milliseconds may appear anywhere in such a capability, enclosed in \$<...> brackets, as in `el=\EK$<3>`, and padding characters are supplied by `tputs` to provide this delay. The delay can be either a number (such as '20'), or a number followed by '*' (such as '3*'). '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of lines affected. This is always 1 unless the terminal has `xenl` and the software uses it.) When '*' is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. Both `\E` and `\e` map to an ESCAPE character, `^x` maps to a control-x for any appropriate x, and the sequences `\n`, `\r`, `\t`, `\b`, `\f`, `\s` gives a new-line, linefeed, return, tab, backspace, formfeed, and space. Other escapes include `\^` for `^`, `\\` for `\`, `\,` for comma, `\:` for `:`, and `\0` for null. (`\0` will produce `\200`, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a `\`.

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second `ind` in the example above.

terminfo

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in terminfo and to build up a description gradually, using partial descriptions with **vi** to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the terminfo file to describe it or bugs in it in **vi**. To easily test a new terminal description, you can set the environment variable **TERMINFO** to a pathname of a directory containing the compiled description you are working on and programs will look there rather than in **/usr/lib/terminfo**. To get the padding for insert line right (if the terminal manufacturer did not document it) a severe test is to edit **/etc/passwd** at 9600 baud, delete 16 or so lines from the middle of the screen, then hit the 'u' key several times quickly. If the terminal messes up, more padding is usually needed. A similar test can be used for insert character.

Basic Capabilities

The number of columns on each line for the terminal is given by the **cols** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **lines** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as TEKTRONIX 4010 series, as well as hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage return, control M.) If there is a code to produce an audible signal (bell, beep, and so on) give this as **bel**.

terminfo

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over (for example, you would not normally use '**cuf1=**' because the space would erase the character moved over).

A very important point here is that the local cursor motions encoded in terminfo are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin**, which have the same semantics as **ind** and **ri** except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion that 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; that is, **am**. If the terminal has a command that moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

terminfo

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

```
33    tty33    tty    model 33 teletype,
      bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,
```

while the Lear Siegler ADM-3 is described as

```
adm3    3    lsi adm3,
      am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H, cud1=^J,
      ind=^J, lines #24,
```

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with **printf** like escapes **%x** in it (see Section 3). For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special **%** codes to manipulate it. Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary.

The **%** encodings have the following meanings:

%%	outputs '%'
%d	print pop() as in printf
%2d	print pop() like %2d
X3d	print pop() like %3d
%02d	
%03d	as in printf
%c	print pop() gives %c
%s	print pop() gives %s
%p[1-9]	push <i>i</i> th parm
%P[a-z]	set variable [a-z] to pop()
%g[a-z]	get variable [a-z] and push it
%c'	char constant c
%{nn}	integer constant nn
%+%-*%/%m	arithmetic (%m is mod): push (pop() op pop())

terminfo

%&% %^	bit operations: push (pop() op pop())
%=%>%<	logical operations; push (pop() op pop())
%!%~	unary operations push (op pop())
%i	add 1 to first two parms (for ANSI terminals)
!? expr %t	if-then-else, %e elsepart is optional.
thenpart %e	else-if's are possible ala Algol 68:
elsepart %;	!? c ₁ %t b ₁ %e c ₂ %t b ₂ %e c ₃ %t b ₃ %e c ₄ %t b ₄ %e %;
	c _i are conditions, b _i are bodies.

Binary operations are in postfix form with the operands in the usual order. That is, to get x-5 you use "%gx%{5}%-".

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cup** capability is
cup=6\E&%p2%2dc%p1%2dY.

The Microterm ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, cup=^T%p1%c%p2%c. Terminals that use %c need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit \n ^D and \r, as the system may change or discard them. (The library routines dealing with terminfo set tty modes so that tabs are never expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus
cup=\E=%p1''%+%c%p2''%+%c. After sending '\E=', this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values) and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

terminfo

If the terminal has row or column absolute cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two parameter sequence (as with the hp2645) and can be used in preference to **cup**. If there are parameterized local motions (for example, move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**, such as the TEKTRONIX 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen), then this can be given as **home**; similarly a fast way of getting to the lower left-hand corner can be given as **ll**; This may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the **\EH** sequence on HP terminals cannot be used for **home**.)

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **Ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines if a true **ed** is not available.)

terminfo

Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line that the cursor is on, then this should be given as **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**. If the terminal has a settable scrolling region (like the vt100), the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is undefined after using this command. It is possible to get the effect of insert or delete line using this command; the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** and **ind** on many terminals without a true insert/delete line, which is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

terminfo

Insert/Delete Character

There are two basic kinds of intelligent terminals (with respect to insert/delete character) that can be described using terminfo. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank, 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; you should give the capability `in`, which stands for insert null. While these are two logically separate attributes (one line vs. multi-line insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

Terminfo can describe both terminals that have an insert mode, and terminals that send a simple sequence to open a blank position on the current line. Give as `smir` the sequence to get into insert mode. Give as `rmir` the sequence to leave insert mode. Now give as `ich1` any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give `ich1`; terminals that send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to `ich1`. Do not give both unless the terminal

terminfo

actually requires both to be used in combination.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence that may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an 'insert mode' and a special code to precede each inserted character, then both **smir/rmir** and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, *n*, will repeat the effects of **ich1** *n* times.

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

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n*, to delete *n* characters, and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as standout mode, representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as **sms0** and **rms0**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many spaces are left.

terminfo

Codes to begin underlining and end underlining can be given as **smul** and **rmul**, respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **blink** (blinking), **bold** (bold or extra bright), **dim** (dim or half_bright), **invis** (blanking or invisible text), **prot** (protected), **rev** (reverse video), **sgr0** (turn off all attribute modes), **smacs** (enter alternate character set mode), and **rmacs** (exit alternate character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attribute is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need be supported by **sgr**, only those for which corresponding separate attribute commands exist.

Terminals with the "magic cookie" glitch (**xmc**) deposit special "cookies" when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the HP2621, 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 new-line, unless the **msg** 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), this can be given as **flash**; it must not move the cursor.

terminfo

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given, which undoes the effects of both of these modes.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the TEKTRONIX 4025, where **smcup** sets the command character to be the one used by terminfo.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1**, **kcuf1**, **kcuu1**, **kcud1**, and **khome**, respectively. If there are function keys such as **f0**, **f1**, ..., **f10**, the codes they send can be given as **kf0**, **kf1**, ..., **kf10**. If these keys have labels other than the default **f0** through **f10**, the labels can be given as **lf0**, **lf1**, ..., **lf10**. The codes transmitted by certain other special keys can be

terminfo

given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all tabs), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character to enter insert mode), **kill1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward/down), **kri** (scroll backward/up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1**, **ka3**, **kb2**, **kc1**, and **kc3**. These keys are useful when the effects of a 3 by 3 directional pad are needed.

Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control I). A "backtab" command, which moves leftward to the next tab stop, can be given as **cbt**. By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs that are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by the **tset** command to determine whether to set the mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the terminfo description can assume that they are properly set.

Other capabilities include **is1**, **is2** and **is3**, initialization strings for the terminal; **iprogram**, the path name of a program to be run to initialize the terminal; and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the terminfo description. They are normally sent to the terminal, by the **tset** program, each time the user logs in. They will be printed in the following order: **is1**, **is2**; setting tabs using **tbc** and **hts**; **if**; running the program **iprogram**; and finally **is**. Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common

terminfo

sequences in **is2** and special cases in **is1** and **is3**. A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is2** and **if**. These strings are output by the **reset** program, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs2** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the vt100 into 80-column mode would normally be part of **is2**, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80 column mode.

If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in **is2** or **if**.

Delays

Certain capabilities control padding in the teletype driver. These are primarily needed by hard copy terminals, and are used by the **tset** program to set teletype modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cupb1**, **ff**, and **tab** will cause the appropriate delay bits to be set in the teletype driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Miscellaneous

If the terminal requires other than null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used.

terminfo

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 that is set to a 23-line scrolling region), the capability **hs** should be given. Special strings to go to the beginning of the status line and to return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The parameter **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to. If escape sequences and other special commands, such as **tab**, work while in the status line, the flag **eslok** can be given. A string that turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status is normally assumed to be the same width as the rest of the screen (**cols**). If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded), the width, in columns, can be indicated with the numeric parameter **wsl**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually control L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters), this can be indicated with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, **tparam** (**repeat_char**, 'x', 10) is the same as "xxxxxxxxxx".

terminfo

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 CENTIX: The environment is to be searched for a **CC** variable, and if found, all occurrences of the prototype character are replaced with the character in the environment variable.

Terminal descriptions that do not represent a specific kind of known terminal, such as switch, dialup, patch, and network, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to virtual terminal descriptions for which the escape sequences are known.)

If the terminal uses xon/xoff handshaking for flow control, give **xon**. Padding information should still be included so that routines can make better decisions about costs, but actual characters will not be transmitted.

If the terminal has a "meta key" that acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm#0** indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the CENTIX virtual terminal protocol, the terminal number can be given as **vt**.

terminfo

Media copy strings that control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen; **mc4**: turn off the printer; and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation **mc5p** takes one parameter and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local; and **pfx** causes the string to be transmitted to the computer.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be cancelled by placing **xx@** to the left of the capability definition, where **xx** is the capability. For example, the entry

```
2621-nl, smkx@, rmkx@, use=2621,
```

defines a 2621-nl that does not have the **smkx** or **rmkx** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

terminfo

Files

`/usr/lib/terminfo/?/*` - files containing terminal descriptions

See Also

curses, **printf** in Section 3; **term** in Section 5.

utmp

Name

utmp, wtmp - utmp and wtmp entry formats

Format

```
#include <sys/types.h>
#include <utmp.h>
```

Description

These files hold user and accounting information for such commands as **who**, **write**, and **login**. Each Application Processor has its own utmp and wtmp files; the two digit AP number is appended to the file name.

The files 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*/
    char ut_line[12]; /*device name (console, lnx)*/
    short ut_pid; /*process id*/
    short ut_type; /*type of entry*/
    struct exit_status {
        short e_termination; /*Proc. terminat. status*/
        short e_exit; /*Process exit status*/
    } ut_exit; /*The exit status of a process
               *marked as DEAD_PROCESS.*/
    time_t ut_time; /*time entry was made*/
};

/*Definitions for ut_type*/
#define EMPTY 0
#define RUN_LVL 1
#define BOOT_TIME 2
#define OLD_TIME 3
#define NEW_TIME 4
#define INIT_PROCESS 5 /*Process spawned by "init"*/
#define LOGIN_PROCESS 6 /*A "getty" process waiting
                        *for login*/
```

utmp

```

#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

```

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

```

See Also

login, who, write in Section 1; getut in Section 3.

Miscellaneous Facilities

intro

Name

intro - introduction to miscellany

Description

This section describes miscellaneous facilities such as macro packages, character set tables, and so on.

environ

Name

environ - user environment

Description

An array of strings called the "environment" is made available by the **exec** system call 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 , time , nice , nohup , and so on, apply in searching for a file known by an incomplete path name. The prefixes are separated by colons(:). login sets PATH=:/bin:/usr/bin .
HOME	Name of the user's login directory, set by login from the password file passwd .
TERM	The kind of terminal for which output is to be prepared. This information is used by commands such as mm , which may exploit special capabilities of that terminal.
TZ	Time zone information. The format is xxx/zzz 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**, or by **exec**. It is unwise to conflict with certain shell variables that are frequently exported by .profile files: **MAIL**, **PS1**, **PS2**, **IFS**.

See Also

env, **login**, **sh** in Section 1; **exec** in Section 2; **getenv** in Section 3; **profile** in Section 4; **term**.

fcntl

Name

fcntl - file control options

Format

```
#include <fcntl.h>
```

Description

The `fcntl` function provides for control over open files. The include file describes *requests* and *arguments* to `fcntl` and `open` (see Section 2).

```
/*Flag values accessible to open and fcntl*/
/*(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*/
#define O_DIRECT 020000 /*perform direct I/O*/
#define O_NODIRECT 040000

/*Flag values accessible only to open*/
#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*/

/*fcntl 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 blocking file locks*/
#define F_SETLK 6 /*Set or clear file locks*/
/*and fall on busy*/
#define F_SETLKW 7 /*Set or clear file locks*/
/*and wait on busy*/
```

fcntl

```
/*file segment locking control structure*/
struct flock {
    short   l_type;
    short   l_whence;
    long    l_start;
    long    l_len;    /*if 0 then until EOF*/
    int     l_pid;    /*returned with F_GETLK*/

/*file segment locking types*/
#define F_RDLCK    01    /*Read lock*/
#define F_WRLCK    02    /*Write lock*/
#define F_UNLCK    03    /*Remove locks*/
```

See Also

`fcntl`, `open` in Section 2.

math

Name

math - math functions and constants

Format

```
#include <math.h>
```

Description

This file contains declarations of all the functions in the Math Library, as well as various functions in the C Library (see Section 3, Library Functions) that return floating-point values.

It defines the structure and constants used by the `matherr` error-handling mechanisms, including the following constant used as an error-return value:

HUGE The maximum value of a single-precision floating-point number.

The following mathematical constants are defined for user convenience:

M_E	The base of natural logarithms (e).
M_LOG2E	The base-2 logarithm of e .
M_LOG10E	The base-10 logarithm of e .
M_LN2	The natural logarithm of 2.
M_LN10	The natural logarithm of 10.
M_PI	The ratio of the circumference of a circle to its diameter. (There are also several fractions of its reciprocal and its square root.)
M_SQRT2	The positive square root of 2.
M_SQRT1_2	The positive square root of 1/2.

For the definitions of various machine-dependent "constants," see the description of the `<values.h>` header file.

math

Files

`/usr/include/math.h`

See Also

`intro`, `matherr` in Section 3; `values`.

modemcap

Name

modemcap - smart modem capability data base

Format

```
/usr/lib/uucp/modemcap
```

Description

Modemcap describes the call placing protocol of smart modems. CENTIX **uucp** and **dial** accept a reference to a modemcap entry in place of an automatic call unit reference in `/usr/lib/uucp/L-devices`. Each entry describes a single modem in a specific configuration.

Modemcap is a text file. Lines that begin with a pound sign (#) are ignored. Other lines make up descriptions.

Each description begins on a new line. The beginning of the description is a list of its names, separated by vertical bars(|). Any of the names, which must not begin with **cu**, can be used in place of the call unit name in `/usr/lib/uucp/L-devices`.

The rest of the description is a list of capabilities, separated by colons(:). If a description extends over more than one line, each line except the last must end with a backslash(\). (The continuation is normally entered as colon-backslash-newline-tab-colon: this produces a single invalid capability, which is ignored.) Here is an example:

```
#bizcomp 1012 - option switch 9 down
bz | bizcomp bizcomp 1012:
:a1=NO ANSWER:b1=NO DIAL TONE:b2=NO ANSWER:c1=1:c2=2:\
:c7=7:d1#1:d5#5:eh=\r:ph=\02D:ps=\02:pw=72:\
:sa=A:sq=Q:sv=V:sx=X:sz=Z:wp=\r:\
:p1=szd5wpd1svwpsqwpsxwpd1phwpc7b1wpc2a1c1b2d1:
```

modemcap

Each capability has three parts:

- 1 The two-character name of the capability.
- 2 A pound sign (#) or equal sign (=). A pound sign indicates a numeric capability. An equal sign indicates a string capability.
- 3 The capability value. For a numeric capability, the value is the number that immediately follows the pound sign. For a string capability, the value is the string of characters, including blanks, between the equal sign and the colon that ends the capability. (If a colon is part of the value, it must be expressed as an octal sequence; see below.) In a string capability, the following sequences stand for single characters:

<code>\xxx</code>	(where <i>xxx</i> is one to three octal digits) The character whose octal value is <i>xxx</i> .
<code>\072</code>	Colon (:).
<code>\200</code>	Null (<code>\000</code> doesn't work).
<code>\E</code>	Escape (<code>\033</code>).
<code>\n</code>	Newline (<code>\012</code>).
<code>\r</code>	Return (<code>\015</code>).
<code>\t</code>	Tab (<code>\011</code>).
<code>\b</code>	Backspace (<code>\010</code>).
<code>\f</code>	Formfeed (<code>\014</code>).
<code>^x</code>	Control- <i>x</i> .

There are four kinds of capabilities: the place call capability, basic features capabilities, the send phone number capability, and send/receive capabilities. Only the place call capability is mandatory.

Place Call Capability

- pl** String capability. Controls the use of the other capabilities. The value of the string is a procedure made up of the other capabilities. A communication program works through **pl**'s value, using each capability as it is encountered; a limited control of execution flow is provided by some special capabilities.

modemcap

Basic Features Capabilities

Basic features capabilities specify strings used to command basic features of the modem. These capabilities never appear in the **pl** value, but are implied by other capabilities. The capability descriptions indicate which capabilities use basic features capabilities and what happens when basic features capabilities are undefined.

ps	Primary command start; string capability. The ps capability specifies the characters that precede modem commands, if required. Used by sx capability.
pe	Primary command end; string capability. The pe capability specifies the characters that must follow modem commands, if required. Used by sx capability.
eh	End phone number; string capability. Used by ph capability.
pa	Pause in phone number; string capability. Used by ph capability.
pw	Pause in phone number and wait for dial tone; string capability. Used by ph capability.

Send Phone Number Capability

ph	String capability. In a single write system call, send a string with three parts: <ol style="list-style-type: none">1 The ph capability's own value.2 The phone number as ASCII digits. Whenever the modem should pause, send the value of the pa capability, if defined. Whenever the modem should pause and wait for a dial tone, send the value of the pw capability, if defined.3 The value of the eh capability, if defined.
-----------	--

modemcap

Send/Receive Capabilities

Send/receive capabilities are different from other capabilities in their naming convention. The first character of the capability name tells the kind of capability. The second character of the name is chosen arbitrarily from the lowercase letters and digits and identifies the particular capability from others of the same kind.

- tx** String capability. Send the value to the modem.
- sx** String capability. In a single write, send a command to the modem. The command has three parts:

- 1 The value of the **ps** capability, if defined.
- 2 The **sx**'s capability's own value.
- 3 The value of the **pe** capability, if defined.

- dx** Numeric capability. Delay for the number of seconds specified in the value.
- wx** String capability; value must be a single character. Wisk through input from modem until the value is read. Put input, up to but not including the terminating character, in the wisk buffer, replacing the previous contents.
- cx** String capability. Compare value with contents of the wisk buffer. Set the comparison flag to EQUAL if they match, NOT_EQUAL otherwise. Do not modify the comparison flag until you execute another **cx**.
- mx** Numeric capability. Skip on EQUAL. If the comparison flag is EQUAL, the next *n* instructions in the **pl** value are skipped, where *n* is the value of **mx**.
- nx** Numeric capability. Skip on NON_EQUAL. If the comparison flag is NOT_EQUAL, the next *n* instructions in the **pl** value are skipped, where *n* is the value of **nx**.
- ax** String capability. Abort on EQUAL. If the comparison flag is EQUAL, abort the phone call. If debug output is specified, print the value of the **ax** capability.
- bx** String capability. Abort on NOT_EQUAL. If the comparison flag is NOT_EQUAL, abort the phone call. If debug output is specified, print the value of the **bx** capability.

modemcap

Example

The Bizcomp 1012 example above assumes that the modem's switch 9 (configuration: TERMINAL/COMPUTER) is down (COMPUTER). With this setting, the modem has the following characteristics:

- Commands to the modem must be preceded by an STX (\002) and followed by a CR (\r). This prevents normal data transmissions from being taken for modem commands.
- The modem's messages to the computer are terse. The following two-character sequences are diagnostics.

1 CR	connection made
2 CR	no connection or no answer
7 CR	dial tone detected

A CR is a command prompt. A communication program that uses the Bizcom 1012 modemcap entry follows the following procedure:

- 1 (szd5wpd1) Send an STX-Z-CR, resetting the modem. Wait five seconds, then read the resulting CR. Wait another one second.
- 2 (svwpsqwpsxwpd1) Send an STX-V-CR (select tone dialing); read the resulting CR. Send an STX-Q-CR (toggle busy detection); read the resulting CR. Send an STX-X-CR (select transparent data mode); read the resulting CR. Wait one second.
- 3 (ph) Send an STX-D, then the phone number. The phone number should include a colon (:) whenever the modem should pause to listen for another dial tone. The description lacks a **pa** capability, so there is no way to pause without waiting for a dial tone.
- 4 (wpc7b1) Read until the next CR. If the input isn't "7," abort with the debug message "NO DIAL TONE."

modemcap

- 5 (wpc2a1c1b2) Read until the next CR. If the input is "2," abort with the debug message "NO ANSWER." Otherwise, if the input isn't "1," abort with the debug message "NO ANSWER."
- 6 (d1) Wait one second. The connection is established.

See Also

dial in Section 3; **uucp** in Section 1.

pilf

Name

pilf, dio - performance improvement in large files and direct I/O

Description

A PILF file system supports the input or output of large amounts of data with a single physical read or write. This requires special strategies for I/O; when standard I/O operations are applied to a PILF file system, it behaves like a standard 1K file system. A PILF file system is created with the **-P** option of **mkfs** (see Section 1).

A file on a PILF file system is allocated by clusters, each of which is equal in size and consists of contiguous blocks. Performance improvement is seen when the DIO (Direct Input/Output) mechanism is used and no read or write crosses a cluster boundary.

A field in the *i*-node determines the file's cluster size. A cluster consists of 2^c 1K blocks, where *c* is the value in the *i*-node. The process that creates a PILF file specifies its cluster size using the **syslocal** system call; if a process has not yet specified a cluster size, the default cluster size, in the superblock, is used. A file's cluster size is determined when it is created; it cannot be changed.

DIO transfers data directly between the process's address space and the disk, bypassing the kernel buffer cache. It is specifically meant to be used on PILF files. DIO is automatically used for reads or writes of multiples of 1K to regular files that are greater than 2K and word aligned.

Caution

A buffer used for DIO must be on an even address. This is the same degree of alignment as a short.

See Also

cp, **mkfs**, **fsck**, **fsdb** in Section 1; **fcntl**, **fork**, **open**, **syslocal** in Section 2; **fs**, **inode** in Section 4; **fcntl**.

prof

Name

prof - profile within a function

Format

```
#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 up to six letters, 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, such as:

```
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.

prof

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 in Section 1; **profil** in Section 2; **monitor** in Section 3.

regex

Name

regex - regular expression compile and match routines

Format

```
#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 <regex.h>

char *compile (instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;
int eof;

int step (string, expbuf)
char *string, *expbuf;

extern char *loc1, *loc2, *locs;

extern int circf, sed, nbra;
```

Description

This page describes general-purpose regular expression matching routines in the form of **ed**, defined in `/usr/include/regex.h`. Programs such as **ed**, **sed**, **grep**, **bs**, **expr**, and so on, 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 <regex.h>"` statement. These macros are used by the **compile** routine.

regexp

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(<i>c</i>)	Cause the argument <i>c</i> to be returned by the next call to GETC() (and PEEKC()). No more than 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(<i>c</i>) is always ignored.
RETURN(<i>pointer</i>)	This macro is used on normal exit of the compile routine. The value of the argument <i>pointer</i> is a pointer to the character after the last character of the compiled regular expression. This is useful to programs that have memory allocation to manage.
ERROR(<i>val</i>)	This is the abnormal return from the compile routine. The argument <i>val</i> is an error number (see Table 5-1, below, for meanings). This call should never return.

Table 5-1 Errors and Meanings

Error	Meaning
11	Range endpoint too large.
16	Bad number.
25	"\digit" out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	\(\) imbalance.
43	Too many \(.
44	More than 2 numbers given in \{ \}.
45	} expected after \.
46	First number exceeds second in \{ \}.
49	[] imbalance.
50	Regular expression overflow.

regexp

The syntax of the **compile** routine is as follows

```
compile (instring, expbuf, endbuf, eof)
```

The first parameter *instring* is never used explicitly by the **compile** routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs that 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 that marks the end of the regular expression. For example, in **ed**, 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 brace(`{`). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point to 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 the **grep** shell command.

regex

There are other functions in this file that perform actual regular expression matching, one of which is the function **step**. The call to **step** is as follows:

```
step(string, expbuf)
```

The first parameter to **step** is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter, *expbuf*, is the compiled regular expression that was obtained by a call of the function **compile**.

The function **step** returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to **step**. The variable set in **step** is *loc1*. This is a pointer to the first character that matched the regular expression. The variable *loc2*, which is set by the function **advance**, points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, *loc1* will point to the first character of string and *loc2* will point to the null at the end of string.

Step uses the external variable *circf* which is set by **compile** if the regular expression begins with \wedge . If this is set, **step** will try to match the regular expression to the beginning of the string only. If more than one regular expression is to be compiled before the first is executed, the value of *circf* should be saved for each compiled expression, and *circf* should be set to that saved value before each call to **step**.

The function **advance** is called from **step** with the same arguments as **step**. The purpose of **step** is to step through the string argument and call **advance** until **advance** returns non-zero, indicating a match, or until the end of string is reached. If you want to constrain *string* to the beginning of the line in all cases, **step** need not be called; simply call **advance**.

regex

When **advance** encounters a `*` or `\{ \}` sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself, trying to match the rest of the string to the rest of the regular expression. As long as there is no match, **advance** will back up along the string until it finds a match or reaches the point in the string that initially matched the `*` or `\{ \}`. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer *locs* is equal to the point in the string at some time during the backing up process, **advance** will break out of the loop that backs up and will return zero. This is used by **ed** and **sed** for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like `s/y*/g` do not loop forever.

The additional external variables *sed* and *nbra* are used for special purposes.

Examples

The following is an example of how the regular expression macros and calls look from the **grep** command:

```
#define INIT      register char *sp = instring;
#define GETC()    (*sp++)
#define PEEKC( )  (*sp)
#define UNGETC(c) (-sp)
#define RETURN(c) return;
#define ERROR(c)  regerr()

#include <regex.h>
...
      (void) compile(*argv, expbuf, &expbuf[ESIZE], '\0');
...
      if (step(linebuf, expbuf))
          succeed( );
```

regex

Files

`/usr/include/regex.h`

Known Problems

The handling of *circf* is kludgy.

The actual code is probably easier to understand than this manual page.

See Also

`bs`, `ed`, `expr`, `grep`, `sed` in Section 1.

stat

Name

stat - data returned by stat system call

Format

```
#include <sys/types.h>
#include <sys/stat.h>
```

Description

The system calls **stat** and **fstat** return data, the structure of which is defined by this include file. The encoding of the field *st_mode* is defined in this file also.

```
/*
 *Structure of the result of stat
 */

struct    stat
{
    dev_t    st_dev;
    ino_t    st_ino;
    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      04000 /*set user id on execution*/

#define S_ISGID      02000 /*set group id on execution*/

#define S_ISVTX      01000 /*save swapped text after use*/
```

stat

```
#define S_IREAD    00400    /*read permission, owner*/
#define S_IWRITE   00200    /*write permission, owner*/
#define S_IEXEC    00100    /*execute/search permission,
                             *owner*/
```

Files

/usr/include/sys/types.h

/usr/include/sys/stat.h

See Also

stat in Section 2; types.

term

Name

term - conventional names for terminals

Description

The names shown in Table 5-2 are used by certain shell commands (for example, `tabs` is maintained as part of the shell environment) in the variable `$TERM`:

Table 5-2 Terminal Names

Name	Description
pt	Burroughs/Convergent Technologies Programmable Terminal
gt	Burroughs/Convergent Technologies Graphics Terminal
freedom	Liberty Freedom 100
1520	Datamedia 1520
1620	DIABLO 1620 and others using the HyType II printer
1620-12	Same as above, in 12-pitch mode
2621	Hewlett-Packard HP2621 series
2631	Hewlett-Packard 2631 line printer
2631-c	Hewlett-Packard 2631 line printer - compressed mode
2631-e	Hewlett-Packard 2631 line printer - expanded mode
2640	Hewlett-Packard 2640 series
2645	Hewlett-Packard HP264n series (other than the 2640 series)
300	DASI/DTC/GSI 300 and others using the HyType I printer
300-12	Same as above, in 12-pitch mode
300s	DASI/DTC/GSI 300s
382	DTC 382
300s-12	Same as above two entries, in 12-pitch mode
3045	Datamedia 3045
33	TELETYPE Model 33 KSR
37	TELETYPE Model 37 KSR
40-2	TELETYPE Model 40/2
40-4	TELETYPE Model 40/4
4540	TELETYPE Model 4540
3270	IBM Model 3270
4000a	Trendata 4000a
4014	TEKTRONIX 4014
43	TELETYPE Model 43 KSR
450	DASI 450 (same as Diablo 1620)
450-12	Same as above, in 12-pitch mode
735	Texas Instruments TI735 and TI725

term

Name	Description
745	Texas Instruments T1745
dumb	Generic name for terminals that lack reverse line-feed and other special escape sequences; likely to work when the real terminal type is not known to the program
sync	Generic name for synchronous TELETYPE 4540-compatible terminals
hp	Hewlett-Packard (same as 2645)
lp	Generic name for a line printer
tn1200	User Electric TermiNet 1200
tn300	User Electric TermiNet 300

Up to 8 characters, chosen from -, a-z, and/or 0-9, make up a basic terminal name. Terminal sub-models and operational modes are distinguished by suffixes beginning with a -. Names should generally be based on original vendor, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name.

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*.

See Also

mm, **sh**, **stty**, **tabs** in Section 1; **profile** in Section 4; **environ**.

types

Name

types - primitive system data types

Format

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

Description

The data types defined in the include file are used in CENTIX code; some data of these types are accessible to user code:

```
typedef struct {int r[1];}*      physadr;
typedef long                  daddr_t;
typedef char*                 caddr_t;
typedef unsigned int          uint;
typedef unsigned short        ushort;
typedef ushort                ino_t;
typedef short                 cnt_t;
typedef long                  time_t;
typedef int                   label_t[13];
typedef short                 dev_t;
typedef long                  off_t;
typedef long                  paddr_t;
typedef long                  key_t;
```

The form *daddr_t* is used for disk addresses except in an i-node on disk. see *fs* in Section 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. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

See Also

fs in Section 4.

values

Name

values - machine-dependent values

Format

```
#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(<i>type</i>)	The number of bits in a specified <i>type</i> (for example, int).
HIBITS	The value of a short integer with only the high-order bit set (in most implementations, 0x8000).
HIBITL	The value of a long integer with only the high-order bit set (in most implementations, 0x80000000).
HIBITI	The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).
MAXSHORT	The maximum value of a signed short integer (in most implementations, 0x7FFF = 32767).
MAXLONG	The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF = 2147483647).
MAXINT	The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).
MAXFLOAT, LN_MAXFLOAT	The maximum value of a single-precision floating-point number, and its natural logarithm.
MAXDOUBLE, LN_MAXDOUBLE	The maximum value of a double-precision floating-point number, and its natural logarithm.

values

MINFLOAT,
LN_MINFLOAT

The minimum positive value of a single-precision floating-point number, and its natural logarithm.

MINDOUBLE,
LN_MINDOUBLE

The minimum positive value of a double-precision floating-point number, and its natural logarithm.

FSIGNIF

The number of significant bits in the mantissa of a single-precision floating-point number.

DSIGNIF

The number of significant bits in the mantissa of a double-precision floating-point number.

Files

`/usr/include/values.h`

See Also

`intro` in Section3; `math`.

varargs

Name

varargs - handle variable argument list

Format

```
#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 the **printf** library function) 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.

varargs

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 the `exec1` system call.

```
#include <varargs.h>
#define MAXARGS    100

/*  exec1 is called by
        exec1(file, arg1, arg2, ..., (char *)0);
*/
exec1(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 execev(file, args);
}
```

Known Problems

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.

varargs

See Also

exec in Section 2; **printf** in Section 3.

Device Files

intro

Name

intro - introduction to device files

Description

This section describes various device files that refer to specific hardware peripherals and CENTIX System device drivers. The names of the entries are generally derived from names for the hardware, as opposed to the names of the files themselves. Characteristics of both the hardware device and the corresponding device driver are discussed where applicable.

To be configured into the CENTIX operating system, each peripheral (or I/O) device must be represented in the overall CENTIX file system by a device file, located in the /dev directory. The contents of a device file point to the device driver, located in the CENTIX kernel, for the device.

When you send data to, for example, a disk, you send the data to the device file in the /dev directory that has been created for that disk. The data, however, is not actually stored in the device file (in the CENTIX file system), but at the disk itself. In the same way, when you load data from a tape, you call it from the device file for the tape device, but the data is actually loaded from the tape itself.

There are two types of CENTIX device files:

- *Block* device files are used for devices that handle I/O data in 1024 bytes (1 kB) blocks. The I/O size is controlled by the operating system's buffer size and is independent of the user's I/O size. Disk and tape devices can be configured as block devices.
- *Character* device files are used for devices that handle raw data streams. The size of I/O transfers in raw data streams are determined either by the software design of the device itself (for terminals and printers) or by the program controlling the device (for disks and tapes).

intro

For those devices that can be used as either block or character, the difference between the two is in performance. One or the other type of device may be necessary for special applications.

With the CENTIX 6.0 system software release, the device file naming conventions for tapes and disks have changed. (Device names for printers and terminals have not changed.) The system now supports both the old and new naming conventions. Old names are linked to the new names internally.

In CENTIX systems before the 6.0 release, the disk devices are named as follows:

```
/dev/[r]xp/ddn
```

where:

- [r] is an optional field that defines the disk as a character — rather than block — device.
- xp is fp if the disk device is connected to an FP; dp if the disk device is connected to a DP.
- dd represents the disk number. CENTIX disk numbers are the same as the BTOS disk device numbers, except that you must add a 0 in front of a one-digit BTOS disk number for CENTIX. That is, if a built-in disk is named d4 in BTOS, dd is 04 in CENTIX. Or, if an SMD disk is named s1 in BTOS, dd is 01 in CENTIX. (Do not add a zero in front of a two-digit BTOS disk number. For BTOS disk s10, dd is 10.)
- n represents the disk partition. Each disk has a maximum of eight partitions (0 through 7).

With the CENTIX 6.0 release, the disk devices on your system are named as follows:

```
/dev/[r]dsk/cndnnsn
```

where:

- [r] is an optional field that defines the disk as a character — rather than block — device.
- cn represents the controller number. The controller number is always c0 if the controller is a file processor (FP). The controller number is always c1 if the controller is a disk processor (DP).

intro

- *dnn* represents the disk number. CENTIX disk numbers are the same as the BTOS disk device numbers, except that you must add a 0 in front of a one-digit BTOS disk number for CENTIX. That is, if a built-in disk is named d4 in BTOS, *nn* is 04 in CENTIX. Or, if an SMD disk is named s1 in BTOS, *nn* is 01 in CENTIX. (Do not add a zero in front of a two-digit BTOS disk number. For BTOS disk s10, *nn* is 10).
- *sn* represents the disk partition. Each disk has a maximum of 8 partitions (0 through 7).

Table 6-1 shows the correlation between the old (pre-6.0 release) and new (6.0 release) naming conventions for built-in disks connected to FPs. Table 6-2 shows the correlation between the old and new naming conventions for storage module device (SMD) drives connected to DPs. Note that in both tables, *n* represents the partition number. Each disk can have up to eight partitions (0 through 7).

Table 6-1 Naming Conventions for Built-In Disk Drives

Pre-6.0 Release	6.0 Release and Later	BTOS Disk Device Name
FIRST FP		
/dev/[r]fp00 <i>n</i>	/dev/[r]dsk/c0d00 <i>sn</i>	d0 [disk cartridge]
/dev/[r]fp01 <i>n</i>	/dev/[r]dsk/c0d01 <i>sn</i>	d1
/dev/[r]fp02 <i>n</i>	/dev/[r]dsk/c0d02 <i>sn</i>	d2
/dev/[r]fp03 <i>n</i>	/dev/[r]dsk/c0d03 <i>sn</i>	d3
SECOND FP		
/dev/[r]fp04 <i>n</i>	/dev/[r]dsk/c0d04 <i>sn</i>	d4
/dev/[r]fp05 <i>n</i>	/dev/[r]dsk/c0d05 <i>sn</i>	d5
/dev/[r]fp06 <i>n</i>	/dev/[r]dsk/c0d06 <i>sn</i>	d6
/dev/[r]fp07 <i>n</i>	/dev/[r]dsk/c0d07 <i>sn</i>	d7
THIRD FP		
/dev/[r]fp08 <i>n</i>	/dev/[r]dsk/c0d08 <i>sn</i>	d8
/dev/[r]fp09 <i>n</i>	/dev/[r]dsk/c0d09 <i>sn</i>	d9
/dev/[r]fp10 <i>n</i>	/dev/[r]dsk/c0d10 <i>sn</i>	d10
/dev/[r]fp11 <i>n</i>	/dev/[r]dsk/c0d11 <i>sn</i>	d11

and so on.

intro

Table 6-2 Naming Conventions for SMD Disk Drives

Pre-60 Release	6.0 Release and Later	BTOS Disk Device Name
FIRST DP		
<i>/dev/[r]dp00n</i>	<i>/dev/[r]dsk/c1d00sn</i>	s0
<i>/dev/[r]dp01n</i>	<i>/dev/[r]dsk/c1d01sn</i>	s1
<i>/dev/[r]dp02n</i>	<i>/dev/[r]dsk/c1d02sn</i>	s2
<i>/dev/[r]dp03n</i>	<i>/dev/[r]dsk/c1d03sn</i>	s3
<i>/dev/[r]dp04n</i>	<i>/dev/[r]dsk/c1d04sn</i>	s4
<i>/dev/[r]dp05n</i>	<i>/dev/[r]dsk/c1d05sn</i>	s5
SECOND DP		
<i>/dev/[r]dp06n</i>	<i>/dev/[r]dsk/c1d06sn</i>	s6
<i>/dev/[r]dp07n</i>	<i>/dev/[r]dsk/c1d07sn</i>	s7
<i>/dev/[r]dp08n</i>	<i>/dev/[r]dsk/c1d08sn</i>	s8
<i>/dev/[r]dp09n</i>	<i>/dev/[r]dsk/c1d09sn</i>	s9
<i>/dev/[r]dp10n</i>	<i>/dev/[r]dsk/c1d10sn</i>	s10
<i>/dev/[r]dp11n</i>	<i>/dev/[r]dsk/c1d11sn</i>	s11
THIRD DP		
<i>/dev/[r]dp12n</i>	<i>/dev/[r]dsk/c1d12sn</i>	s12
<i>/dev/[r]dp13n</i>	<i>/dev/[r]dsk/c1d13sn</i>	s13
<i>/dev/[r]dp14n</i>	<i>/dev/[r]dsk/c1d14sn</i>	s14
<i>/dev/[r]dp15n</i>	<i>/dev/[r]dsk/c1d15sn</i>	s15
<i>/dev/[r]dp16n</i>	<i>/dev/[r]dsk/c1d16sn</i>	s16
<i>/dev/[r]dp17n</i>	<i>/dev/[r]dsk/c1d17sn</i>	s17

and so on.

With the CENTIX 6.0 release, the conventions for naming tape drives have also changed.

In CENTIX systems before the 6.0 release, the tape drives are named as follows:

```
/dev/[n][r]mntn
```

where:

- [n] indicates that the tape is not to rewind a tape file closes. The default is that the tape automatically rewinds.
- [r] indicates that the tape device will handle raw data streams rather than one kB blocks of data.

intro

- *n* represents the tape drive in the system. *n* is 0 for the first half-inch tape drive on the system, 1 for a quarter-inch cartridge (QIC) tape drive, 2 for the second half inch tape drive on the system, 3 for the third, and so on.

With the 6.0 release, the tape drives on your system are named as follows:

```
/dev/[r]mt/cndn[n]
```

where:

- [r] indicates that the tape device will handle raw data streams rather than one kB blocks of data.
- *cn* represents the controller number. For a QIC tape drive, *cn* is always 0. For a half-inch tape drive, *cn* is always 1.
- *dn* represents the tape drive on the controller. You can have only one QIC drive on your system; it is d0. The first half inch tape drive is d0, the second is d1, and so on.
- [n] indicates that the tape is not to rewind when a tape file closes. The default is that the tape automatically rewinds.

Table 6-3 Naming Conventions for Tape Drives

	Pre-6.0 Release	6.0 Release and Later
First half-inch drive	/dev/[n][r]mt0	/dev/[r]mt/c1d0[n]
QIC drive	/dev/[n][r]mt1	/dev/[r]mt/c0d0[n]
Second half-inch tape drive	/dev/[n][r]mt2	/dev/[r]mt/c1d1[n]
Third half-inch tape drive	/dev/[n][r]mt3	/dev/[r]mt/c1d2[n]
and so on.		

console

Name

console - console terminal

Description

The special file `console` designates a standard destination for system diagnostics. The kernel writes its diagnostics to this file, as does any user process with messages of system-wide importance. If `console` is associated with a physical terminal, then console messages also appear on that terminal; it is not necessary to have `console` associated with a physical terminal.

Note that `inittab` (see Section 4) does not normally post a `getty` process on `console`. This is because `console` might become associated with a terminal that is already a login terminal. Each Application Processor has its own `console`, which can be associated with any terminal or with no terminal at all. Whether or not the `console` is associated with a terminal, the most recent console output is saved in a circular buffer.

I/O operations on `console` by a process running on an AP affect the `console` for that AP. The exact meaning depends on whether or not the `console` is associated with a terminal.

- If the `console` is associated with a terminal, all I/O operations to `console`, including `ioctl` system calls, have the same affect as if applied directly to the terminal, except that the output is duplicated on the console buffer.
- If the `console` is not associated with a terminal, all attempts to read the `console` return an end of file condition, all writes to the `console` go only to the console buffer, and `ioctl` operations have no effect on any terminal.

If the kernel debugger is enabled, a `CODE-b` on the terminal associated with the `console` activates the kernel debugger. The command `go` to the kernel debugger resumes normal processing.

The `console` shell command and `syslocal` system calls control terminal association and print the buffers of AP consoles.

console

Files

`/dev/console`

Caution

The kernel debugger is not a supported product and may disappear without warning. Normal system processing is suspended while the kernel debugger is active.

See Also

`console` in Section 1: `syslocal` in Section 2.

dsk

Name

dsk - winchester, cartridge, and floppy disks

Description

The files `/dev/[r]dsk/cndnnsn` refer to slices on winchester, cartridge, and floppy disks, where:

- `[r]` is an optional field that you include when you are loading the file system to a raw memory device. A device that is defined as raw handles raw data streams (one character at a time) rather than one kB blocks of data.
- `cn` represents the controller number. The controller number is always `c0` if the controller is a file processor (FP). The controller number is always `c1` if the controller is a disk processor (DP).
- `dnn` represents the disk number. CENTIX disk numbers are the same as the BTOS disk sevice numbers, except that you must add a 0 in front of a one-digit BTOS disk number for CENTIX. That is, if a built-in disk is named `d4` in BTOS, `nn` is `04` in CENTIX. Or, if an SMD disk is named `s1` in BTOS, `nn` is `01` in CENTIX. (Do not add a zero in front of a two-digit BTOS disk number. For BTOS disk `s10`, `nn` is `10`).
- `sn` represents the disk partition. Each disk has a maximum of 8 partitions (0 through 7).

In the XE 500 CENTIX System architecture, BTOS manages disk initialization and low-level input/output; CENTIX only accesses the disks to store and retrieve data. A disk special file is a reference to a BTOS disk file set aside specially for CENTIX's use. The BTOS file is called a file system partition and is created using the `crup` shell command (see Section 1). The relationship between file system partitions and CENTIX special files is controlled by the BTOS file system configuration file, `[Sys]<Sys>ConfigUFS.sys`. For more information on using disk devices, see the *XE 500 CENTIX Administration Guide*.

dsk

Files

/dev/dsk/
/dev/rdisk/
/dev/dump?
/dev/boot?

See Also

crup, **mknod**, **ofcopy** in Section 1; **ioctl** in Section 2; **intro**.

fp

Name

fp - winchester, cartridge, and floppy disks

Description

This entry describes disk device naming conventions prior to the CENTIX 6.0 release. It is included for compatibility with earlier versions of CENTIX. If your CENTIX system is release 6.0 or later, refer to the entry for **dsk**, earlier in this section.

The files `/dev/fp000` through `/dev/fp64n` and `rfp000` through `rfp64n` refer to slices on winchester, cartridge, and floppy disks. An `r` in the name indicates the character (raw) interface. The three hexadecimal digits are the file processor number, disk number, and slice number. The cartridge drive is disk 0 on file processor 0.

XE 500 CENTIX System architecture greatly simplifies the CENTIX disk interface: BTOS manages disk initialization and low-level input/output; CENTIX only accesses the disks to store and retrieve data. A disk special file is a reference to a BTOS disk file set aside specially for use by CENTIX. The BTOS file is called a file system partition and is created by the **crup** shell command. The relationship between file system partitions and CENTIX special files is controlled by the BTOS file system configuration file, `[Sys]<Sys>ConfigUFS.sys`.

See Also

crup, **mknod**, **ofcopy** in Section 1; **ioctl** in Section 2.

lp

Name

lp - parallel printer interface

Description

Lp is an interface to the parallel printer channel. Bytes written are sent to the printer. Opening and closing produce page ejects. Unlike the serial interfaces (**termio**), the lp driver never prepends a carriage return to a new line (line feed). The lp driver does have options to filter output, for the benefit of printers with special requirement. The driver also controls page format. Page format and filter options are controlled with the **ioctl** system call:

```
#include <sys/lprio.h>
ioctl (filides, command, arg)
```

where *command* is one of the following constants:

LPRSET Set the current page format from the location pointed to by *arg*; this location is a structure of type **lprio**, declared in the header file:

```
struct lprio {
    short ind;
    short col;
    short line;
}
```

Arg should be declared as follows:

```
struct lprio *arg;
```

Ind is the page indent in columns, initially 4. *Col* is the number of columns in a line, initially 132. *Line* is the number lines on a page, initially 66. A new-line that extends over the end of a page is output as a formfeed. Lines longer than the line length minus the indent are truncated.

LPRGET Get the current page format and put it in the **lprio** structure pointed to by *arg*.

lp

LPRSOPTS

Set the filter options from *arg*, which must be of type int. *Arg* should be the logical or of one or more of the following constants, defined in the header file:

Constant	Value	Meaning
LPNOBS	4	No backspace. Set this bit if the printer cannot properly interpret backspace characters. The driver uses carriage return to produce equivalent overstriking.
LPRAW	8	Raw output. Set this bit if the driver must not edit output in any way. The driver ignores all other option bits in the minor device number.
LPCAP	16	Capitals. This option supports printers with a "half-ASCII" character set. Lowercase is translated to uppercase.
LPNOCR	32	No Carriage Return. This option supports printers that do not respond to a carriage return (character OD hexadecimal). Carriage returns are changed to new-lines. If No Newline is also set, carriage returns are changed to form feeds.
LPNOFF	64	No Form Feed. This option supports printers that do not respond to a form feed (character OC hexadecimal). Form feeds are changed to new-lines. If No Newline is also set, form feeds are changed to carriage returns.

lp

LPNON	12	No Newline. This option supports printers that do not respond to a new-line (character 0A hexadecimal). New-lines are changed to carriage returns. If No Carriage Return is also set, new-lines are changed to form feeds.
-------	----	--

Setting all three of No Carriage Return, No Newline, and No Form Feed has the same effect as setting none of them.

LPRGOPTS

Get the current state of the filter options and put them in *arg*, which must be an int.

Files

/dev/lp

See Also

lpr, lpset in Section 1.

mem

Name

mem, kmem - core memory

Description

Mem is a special file that is an image of the core memory of the CENTIX-based processor board. It may be used, for example, to examine, and even to patch the system.

Byte addresses in mem are interpreted as memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file kmem is the same as mem except that kernel virtual memory rather than physical memory is accessed.

Caution

When reading and writing memory in other processes, reads and writes are done in multiples of 1K. As a result, the data may actually change between 1K reads and writes.

Files

/dev/memxx, /dev/kmemxx, where xx is the two-digit processor number.

mt

Name

mt - interface for magnetic tape

Description

This interface provides access to all magnetic tape drives.

`mtx` is the block device with rewind on close for drive `x`. To get the no-rewind device, prepend `n`; to get the raw (character) device, prepend `r`; and to get the no-rewind on close, raw device, prepend `nr`.

There can be up to four drives, any of which can be built-in quarter-inch cartridge (QIC) drives or external drives controlled by a Storage Processor. The connection between drives and drive numbers is in the file system configuration file, under BTOS.

Tape files are separated by tape marks, also known as EOFs. Closing a file open for writing writes one tape mark on a QIC drive and two tape marks on other drives; if the device was no-rewind, the tape is left positioned just after the single QIC tape mark or between the two marks. If the file was a no-rewind file, reopening the drive for writing overwrites the second mark, if there is one, and creates another tape file. Thus on a QIC drive, a single tape mark separates the tape files and ends the tape; on other drives, a single tape mark separates the tape files and a double mark ends the tape.

Here are summaries of block and character device features:

- The block devices read and write only 1024-byte physical blocks; reads and writes of other sizes are resolved into 1K physical I/O. Seeks are ignored on QIC drives. On other drives seeks are allowed, but once the file is opened, reading is restricted to between the last write and the next tape mark. Reading the tape mark produces a zero-length read and leaves the tape positioned after the tape mark; if the file is a no-rewind file, the program can access the next tape file by closing the device and then reopening or opening another device for the same drive.

mt

- On the raw devices, each read or write reads or writes the next physical block. A read must match the size of a normal tape block. The size of a write determines the size of the next block; Write sizes must be a multiple of 512 on QIC drives, a multiple of 2 on other drives. Read/write buffers must begin on an even address; this is the same alignment as **short**. Seeks are ignored. Reading a tape mark produces a zero-length read and leaves the tape positioned after the mark; the program can, without closing the device, read the next tape file.

Files

```
/dev/mt/*  
/dev/nmt/*  
/dev/rmt/*  
/dev/nrmt/*
```

Caution

A nondata error cannot be recovered from except by closing the device.

A QIC tape has no special mark for end of tape, as opposed to end of file.

null

Name

null - the null file

Description

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

Files

`/dev/null`

prf

Name

prf - operating system profiler

Description

The **prf** file provides access to activity information in the operating system. Writing the file loads the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file **prf** is a pseudo-device with no associated hardware.

Files

`/dev/prf`

See Also

profiler in Section 1.

termio

Name

termio - general terminal interface

Description

CENTIX systems use a single interface convention for all RS-232 and cluster (RS-422) terminals, although cluster terminals do not use all the features of the convention. The convention is almost completely taken from the UNIX System V interface for asynchronous terminals.

Two kinds of terminals use this convention:

- RS-232 terminals connected to channels on the XE 500 itself.
- PT 1500 cluster terminals. Generally a cluster channel supports more than one PT 1500; some terminals are indirectly connected through other terminals. Cluster terminals use the same interface as directly connected RS-232 terminals, except that hardware control operations are meaningless on cluster terminals. (Note that "cluster terminal" refers to the way the terminal is used, not to the terminal itself; a PT 1500 terminal can serve as an RS-232 terminal or as a cluster terminal.)

A single naming convention applies to regular RS-232 and cluster terminals. A direct RS-232 or cluster terminal has a name of the form `ttyxxx`, where `xxx` is the terminal's number expressed in three digits.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files; they are opened by `getty` and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the control terminal for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a `fork` system call. A process can break this association by changing its process group using the `setpgrp` system call.

termio

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, the character generated by a PT 1500 BACKSPACE key (ASCII BS, Control-H on most terminals) erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character @ kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a keystroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The erase and kill characters may be changed.

termio

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR	(Rubout of ASCII DEL; generated by a PT 1500 DELETE key) generates an interrupt signal that is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see signal in Section 2.
QUIT	(Control- <code> </code> or ASCII FS; generated by a PT 1500 CODE-CANCEL key) generates a quit signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called core) will be created in the current working directory.
ERASE	(Control-H or ASCII BS; generated by a PT 1500 BACKSPACE key) erases the preceding character. It will not erase beyond the start of a line, as delimited by an NL, EOF, or EOL character.
KILL	(<code>@</code>) deletes the entire line, as delimited by an NL, EOF, or EOL character.
EOF	(Control-D or ASCII EOT; generated by a PT 1500 FINISH key) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.
NL	(ASCII LF) is the normal line delimiter. It cannot be changed or escaped.
EOL	(ASCII NUL) is an additional line delimiter, like NL. It is not normally used.
STOP	(Control-S or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.
START	(Control-Q or ASCII DC1) is used to resume output that has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters cannot be changed or escaped.

termio

The character values for INTR, QUIT, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a hangup signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any subsequent read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Several `ioctl` system calls apply to terminal files. The primary calls use the following structure, defined in `<termio.h>`:

```
#define NCC 8
struct termio {
    unsigned short c_iflag; /*input modes*/
    unsigned short c_oflag; /*output modes*/
    unsigned short c_cflag; /*control modes*/
    unsigned short c_lflag; /*local modes*/
    char c_line; /*line discipline*/
    unsigned char c_cc[NCC]; /*control chars*/
};
```

termio

The special control characters are defined by the array *c_cc*. The relative positions for each function are as follows:

0	INTR
1	QUIT
2	ERASE
3	KILL
4	EOF
5	EOL
6	reserved
7	reserved

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map upper-case to lower-case on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise, if BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three character sequence: 0377, O, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

termio

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR character is translated into an NL character.

If IUCLC is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character will restart output that has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is all bits clear.

The *c_oflag* field specifies the system treatment of output.

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	

termio

TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form-feed vertical-tab delay is specified, it lasts for about 2 seconds.

termio

New-line delay lasts about 0.10 seconds. If `ONLRET` is set, the carriage-return delays are used instead of the new-line delays. If `OFILL` is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.18 seconds. If `OFILL` is set, delay type 1 transmits one or two fill characters, and type 2 and 3, two fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.04 seconds. Type 3 specifies that tabs are to be expanded into spaces. If `OFILL` is set, delay type 1 transmits zero or two fill characters and delay type 2 transmits 1 fill character.

Backspace delay lasts about 0.05 seconds. If `OFILL` is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The `c_cflag` field describes the hardware control of the terminal (not used on cluster terminals):

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud

termio

B4800	0000014	4800 baud
B9600	0000015	9600 baud
EXTA	0000016	19200 baud
EXTB	0000017	External clock.
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored. EXTB specifies external clocking.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stop bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

termio

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise modem control is assumed.

The initial hardware control value after open is B9600, CS8, CREAD, HUPCL.

The *c_iflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (O) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (for example, 0377).

termio

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

<i>for.</i>	<i>use.</i>
.	\\.
	\\
~	\\~
{	\\{
}	\\}
\	\\

For example, A is input as \a, \n as \\n, and \N as \\N.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex).

termio

Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The primary **ioctl** system calls have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are:

TCGETA	Get the parameters associated with the terminal and store in the termio structure referenced by <i>arg</i> .
TCSETA	Set the parameters associated with the terminal from the structure referenced by <i>arg</i> . The change is immediate.
TCSETAW	Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.
TCSETAF	Wait for the output to drain, then flush the input queue and set the new parameters.

Additional **ioctl** calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK	Wait for the output to drain. If <i>arg</i> is 0, then send a break (zero bits to 0.25 seconds).
TCXONC	Start/stop control. If <i>arg</i> is 0, suspend output; if 1, restart suspended output; if 2, transmit XOFF; if 3, transmit XON.
TCFLSH	If <i>arg</i> is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

termio

Files

`/dev/tty???` `/dev/tp????`

Caution

The default value for ERASE is backspace rather than the historical #.

Known problems

Local RS-232 terminals do not currently provide hangup (B0), draining, flushing, or delay.

See Also

`stty`, `ioctl` in Section 2; `tp`, `tty`.

tp

Name

tp - controlling terminal's local RS-232 channels

Description

The tp devices access the RS-232 channels on the controlling terminal. The terminal must be a cluster terminal configured to permit use of the local RS-232 channels (see **termio**). Just as `/dev/tty` permits a process to conveniently access its process group's controlling terminal (see **tty**), `/dev/tp1` and `/dev/tp2` access the controlling terminal's RS-232 channels without reference to the terminal number. This is convenient for accessing the user's local hardware, such as a telephone with an RS-232 interface.

See Also

tty.

tty

Name

tty - controlling terminal interface

Description

The file `/dev/tty` is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

If the terminal is under window management, a process group is controlled by a specific window, and I/O on `/dev/tty` is directed to that window. A terminal can control one process group in each window. See **window**.

Files

`/dev/tty`

See Also

tp, **window**.

window

Name

window - window management primitives

Format

```
#include <sys/window.h>
```

Description

Window management (see **wm** in Section 1) provides a superset of windowless terminal features. This entry describes terminal file features special to window management. Window management features are designed not to interfere with programs that do not know about window management. Such design includes simple extensions to the CENTIX System's standard concepts of file descriptor and control terminal.

- Each terminal file descriptor has an associated window number, a small positive integer that identifies a window. A window number is the most primitive way to refer to a window, and should not be confused with the window ID used by window management sub-routines. A new window gets the smallest window number not already in use. Closing a window frees its number for possible assignment to a later window. Output and control calls on the file descriptor apply only to the descriptor's window; input calls succeed only when the window is active.

A file descriptor created by a **dup** system call or inherited across a **fork** system call inherits the original descriptor's window number. All the file descriptors in such a chain of inheritance, provided they belong to processes in the same process group, are affected when **ioctl** changes the window number of any of them.

window

- When a process group's control terminal is under window management, the process group is actually controlled by a particular window. Such can have more than one process group, each controlled by a different window. Keyboard-generated signals (interrupt and quit) go to the process group controlled by the active window.

When the user creates a new window by using the SPLIT key, the window manager forks a process for that window. The new process inherits file descriptors for standard input (0), standard output (1), and standard error (2) that are associated with the new window. The new process is leader of a process group controlled by the new window.

Programs that create and use windows use window management `ioctl` calls. Such calls take the form

```
ioctl (fildes, command, arg)
struct wiocctl *arg;
```

Fildes is a file descriptor for terminal and window affected, *command* is a window management command (see below), *arg* is a pointer to the following structure, declared in `<sys/window.h>`:

```
#define NWCC 2

struct wiocctl {
    wndw_t          wi_dflltwn dw;
    wndw_t          wi_wndw;
    slot_t wi_mycpu slot;
    slot_t wi_destcpu slot;
    port_t wi_bport;
    char wi_dummy;
    unsigned char wi_cc[NWCC];
};
```

Window management `ioctl` calls `get (WIOCGET)` and `set (WIOCSET and WIOCSETP)` terminal attributes described in the `wiocctl` structure:

<code>wi_dflltwn dw</code>	The window number for the process's default window. If the process does an <code>open</code> on <code>/dev/tty</code> , the new file descriptor is associated with the default window.
----------------------------	--

window

<code>wi_wndw</code>	The window number for the window that <i>fildev</i> (<code>ioctl</code> 's first parameter) is associated with.
<code>wi_mycpuslot</code>	The slot number of the process's host processor. (Not settable.)
<code>wi_destcpuslot</code>	The slot number of the processor that drives the terminal. (Not settable.)
<code>wi_bport</code>	The terminal's Cluster Processor or Terminal Processor channel number. (Not settable.)
<code>wi_cc</code>	Not used by the CENTIX kernel. A value supplied by a <code>WIOCSET</code> or <code>WIOCSETP</code> is stored in a place associated with window <code>wi_wndw</code> . A subsequent <code>WIOCGET</code> on the same window retrieves the information.

Here are the window management `ioctl` commands:

<code>WIOCGET</code>	Get information on calling process and file descriptor <i>fildev</i> . Fill in <i>arg</i> .
<code>WIOCSET</code>	Set values for calling process and file descriptor <i>fildev</i> from information in <i>arg</i> . Has no effect on process group-control terminal relationship.
<code>WIOCSETP</code>	Set values for calling process and file descriptor <i>fildev</i> from information in <i>arg</i> . The window specified in <i>arg</i> -> <code>wi_wndw</code> becomes the process's group's controlling terminal provided the following: <ul style="list-style-type: none"> The calling process is the process group leader. The process group is not currently controlled by another window on this or any other terminal. The specified window is not already a control window.
<code>WIOCCLRP</code>	Only valid executed by process group leader. The process group ceases to have a control terminal or window and the control terminal/window ceases to control any process group. The process group is free to find another control terminal/window, and the old control terminal/window is free to become the control terminal/window for another process group.

window

WIOCCLUSTER	ioctl returns 1 if and only if the terminal is a cluster terminal.
WIODIRECT	Enable direct sending of terminal IPC requests.
WIOCUNDIRECT	Disable direct sending of terminal IPC requests.

An **open** on a terminal special file other than `/dev/tty` (for example, `/dev/tty000`) produces a file descriptor for the lowest-numbered open window. **ioctl** can move this file descriptor to any window.

An **open** can also obtain a controlling terminal/window. The requirements are the same as for **WIOCSETP**.

Files

`/dev/tty` - control terminal
`/dev/tty???` - terminals

Cautions

WIODIRECT and WIOCUNDIRECT are required by the operating system. Their use by user programs is inadvisable.

Use these features in as standard and conservative a way as possible. The best way to enforce standards is to use window management through the library calls described in Section 3.

See Also

stty, **wm** in Section 1; **dup**, **fork**, **ioctl**, **open** in Section 2; **wmgetid**, **wmlayout**, **wmop**, **wmsetid** in Section 3; **termio**, **tty**.

Permuted Index

This index includes entries for all pages of all four volumes of this guide. The entries themselves are based on the one-line descriptions or titles found in the **Name** portion of each manual entry; the significant words (keywords) of these descriptions are listed alphabetically down the center of the index.

The permuted index is a keyword-in-context index that has three columns. To use the index, read the center column to look up specific commands by name or by subject topics. Note that the entry may begin in the left column or wrap around and continue into the left column. A period (.) marks the end of the entry, and a slash (/) indicates where the entry is continued or truncated. The right column gives the manual entry under which the command or subject is described; following each manual entry name is the section number, in parentheses, in which that entry can be found.

/lto3: convert between	3-byte integers and long/	l3tol(3)
comparison. diff3:	3-way differential file	diff3(1)
between long integer/	a64l, l64a: convert	a64l(3)
/obtain and	abandon exchanges.	exchanges(2)
fault.	abort: generate an IOT	abort(3)
absolute value.	abs: return integer	abs(3)
adb:	absolute debugger	adb(1)
abs: return integer	absolute value.	abs(3)
ceiling, remainder,	absolute value/ /floor,	floor(3)
allow/prevent LP/	accept, reject:	accept(1)
times of/ touch: update	access and modification	touch(1)
times. utime: set file	access and modification	utime(2)
/ofCloseAllFiles:	Access BTOS files	ofopenfile(3)
accessibility of a/	access: determine	access(2)
in a/ sputl, sgetl:	access long integer data	sputl(3)

Index-2

sadp: disk	access profiler.	sadp(1)
common object file	access routines. ldfcn:	ldfcn(4)
file systems for optimal	access time. /copy	dcopy(1)
locking: exclusive	access to regions of a/	locking(2)
/endutent, utmpname:	access utmp file entry.	getut(3)
access: determine	accessibility of a file.	access(2)
or disable process	accounting. /enable	acct(2)
/manipulate connect	accounting records.	fwtmp(1)
process accounting.	acct: enable or disable	acct(2)
sin, cos, tan, asin,	acos, atan, atan2:/	trig(3)
killall: kill all	active processes.	killall(1)
sag: system	activity graph.	sag(1)
sa1, sa2, sadc: system	activity report package.	sar(1)
File Processor system	activity reporter.	fpsar(1)
sar: system	activity reporter.	sar(1)
SCCS file editing	activity. /print current	sact(1)
process data and system	activity. /report	timex(1)
	adb: absolute debugger	adb(1)
BTOS queue. quAdd:	add a new entry to a	quadd(3)
putenv: change or	add value to/	putenv(3)
administer SCCS files.	admin: create and	admin(1)
admin: create and	administer SCCS files.	admin(1)
alarm: set a process	alarm clock.	alarm(2)
alarm clock.	alarm: set a process	alarm(2)

for sendmail.	aliases: aliases file	aliases(5)
sendmail. aliases:	aliases file for	aliases(5)
/ofDelete:	allocate BTOS files.	ofcreate(3)
data segment space	allocation. /change	brk(2)
calloc: main memory	allocator. /realloc,	malloc(3)
fast main memory	allocator. /mallinfo:	malloc(3) (fast version)
accept, reject:	allow/prevent LP/	accept(1)
brc, bcheckrc, rc,	allrc, conrc: system/	brc(1)
running process/ renice:	alter priority of	renice(1)
sort: sort	and/or merge files.	sort(1)
and link editor output.	a.out: common assembler	a.out(4)
Processor number.	apnum: print Application	apnum(1)
number. apnum: print	Application Processor	apnum(1)
console: control	Application Processor/	console(1)
/a process on a specific	Application Processor.	spawn(1)
/a process on a specific	Application Processor.	spawn(3)
/to commands and	application programs.	intro(1)
code. exServeRq:	appropriate a request	exserverq(2)
maintainer for portable/	ar: archive and library	ar(1)
format.	ar: common archive file	ar(4)
arithmetic/ bc:	arbitrary-precision	bc(1)
maintainer for/ ar:	archive and library	ar(1)
cpio: format of cpio	archive	cpio(4)
ar: common	archive file format.	ar(4)

Index-4

header of a member of an	archive file. /archive	ldahread(3)
/convert object and	archive files to common/	convert(1)
ldahread: read the	archive header of a/	ldahread(3)
tar: tape file	archiver.	tar(1)
maintainer for portable	archives. /and library	ar(1)
cpio: copy file	archives in and out.	cpio(1)
varargs: handle variable	argument list.	varargs(5)
/output of a varargs	argument list.	vprintf(3)
xargs: construct	argument list(s) and/	xargs(1)
/get option letter from	argument vector.	getopt(3)
expr: evaluate	arguments as an/	expr(1)
echo: echo	arguments.	echo(1)
bc: arbitrary-precision	arithmetic language.	bc(1)
expr: evaluate arguments	as an expression.	expr(1)
	as: assembler.	as(1)
ascii: map of	ASCII character set.	ascii(5)
hd: hexadecimal and	ascii file dump.	hd(1)
character set.	ascii: map of ASCII	ascii(5)
long integer and base-64	ASCII string. /between	a64l(3)
atof: convert	ASCII string to/	atof(3)
date/ /localtime,	asctime, tzset: convert	ctime(3)
gmtime,		
sin, cos, tan,	asin, acos, atan, atan2:/	trig(3)
help:	ask for help.	help(1)

editor/ a.out: common	assembler and link	a.out(4)
as:	assembler.	as(1)
assertion.	assert: verify program	assert(3)
assert: verify program	assertion.	assert(3)
setbuf, setvbuf:	assign buffering to a/	setbuf(3)
wmsetid, wmsetids:	associate a file/	wmsetid(3)
commands at a later/	at, batch: execute	at(1)
cos, tan, asin, acos,	atan, atan2: sin,	trig(3)
tan, asin, acos, atan,	atan2: trigonometric/	trig(3)
string to/	atof: convert ASCII	atof(3)
strtod,	atof: convert string to/	strtod(3)
integer. strtol, atol,	atoi: convert string to	strtol(3)
string to/ strtol,	atol, atoi: convert	strtol(3)
process. wait:	await completion of	wait(1)
and processing/	awk: pattern scanning	awk(1)
request. qurmove: take	back a BTOS queue	qurmove(3)
ungetc: push character	back into input stream.	ungetc(3)
finc: fast incremental	backup.	finc(1)
recover files from a	backup tape. frec:	frec(1)
	banner: make posters.	banner(1)
modem capability data	base. modemcap: smart	modemcap(5)
terminal capability data	base. termcap:	termcap(4)
terminal capability data	base. terminfo:	terminfo(4)

Index-6

between long integer and (visual) display editor	base-64 ASCII string. /convert	a64l(3)
portions of path names.	based on ex. /screen-oriented	vi(1)
at a later time. at, arithmetic language. system initialization/ copy.	basename, dirname: deliver	basename(1)
cb: C program	batch: execute commands	at(1)
j0, j1, jn, y0, y1, yn:	bc: arbitrary-precision	bc(1)
/install object files in fread, fwrite:	bcheckrc, rc, allrc, conrc:	brc(1)
bsearch:	bcopy: interactive block copy.	bcopy(1)
tfind, tdelete, twalk: manage	bdiff: big diff.	bdiff(1)
bcopy: interactive	beautifier.	cb(1)
sum: print checksum and sync: update the super	Bessel functions.	bessel(3)
df: report number of free disk	bfs: big file scanner.	bfs(1)
conrc: system initialization/ spare allocation.	binary directories.	cpset(1)
	binary input/output.	fread(3)
	binary search a sorted table.	bsearch(3)
	binary search trees, tsearch,	tsearch(3)
	block copy.	bcopy(1)
	block count of a file	sum(1)
	block.	sync(1)
	blocks.	df(1)
	brc, bcheckrc, rc, allrc,	brc(1)
	brk, sbrk: change data segment	brk(2)

compiler/interp reter/	bs: a	bs(1)
sorted table	bsearch: binary search a	bsearch(3)
/ofDIDir, ofReadDirSector:	BTOS directory functions.	ofdir(3)
ofWrite: Input/output on a	BTOS file. ofRead,	ofread(3)
ofRename: rename a	BTOS file.	ofrename(3)
ofSetFileStatus:	BTOS File Status.	ofstatus(3)
ofcopy: copy to or from the directories. ofls: list	BTOS file system.	ofcopy(1)
/ofDelete: Allocate	BTOS files and	ofls(1)
ofed, ofvi: edit	BTOS files.	ofcreate(3)
ofCloseAllFiles: Access	BTOS files.	ofeditors(1)
interpreter for interactive	BTOS files. /ofCloseFile.	ofopenfile(3)
CENTIX kernel and copy it to	BTOS JCL. ofcli: command line	ofcli(1)
quAdd: add a new entry to a	BTOS. mkboot: reformat	mkboot(1)
quReadKeyed: examine	BTOS queue.	quadd(3)
quRemove: take back a	BTOS queue. quReadNext.	quread(3)
stdio: standard	BTOS queue request.	quremove(3)
setbuf, setvbuf: assign	buffered input/output package.	stdio(3)
mknod:	buffering to a stream	setbuf(3)
swapshort, swaplong: translate	build special file.	mknod(1)
swab: swap	byte orders to Motorola/Intel.	swapshort(3)
	bytes.	swab(3)

Index-8

cc:	C compiler.	cc(1)
cflow: generate	C flowgraph.	cflow(1)
cpp: the	C language preprocessor.	cpp(1)
cb:	C program beautifier.	cb(1)
lint: a	C program checker.	lint(1)
cxref: generate	C program cross reference.	cxref(1)
ctrace:	C program debugger.	ctrace(1)
	cal: print calendar.	cal(1)
dc: desk	calculator.	dc(1)
cal:print	calendar.	cal(1)
service.	calendar: reminder	calendar(1)
cú:	call another computer system.	cu(1)
data returned by stat system	call. stat:	stat(5)
malloc, free, realloc,	calloc: main memory allocator.	malloc(3)
fast/ malloc, free, realloc,	calloc, mallopt, mallinfo:	malloc(3) (fast version)
intro: introduction to system	calls and error number.	intro(2)
link and unlink system	calls. link, unlink: exercise	link(1)
to an LP line printer. lp.,	cancel: send/cancel requests	lp(1)
modemcap: smart modem	capability data base.	modemcap(5)
termcap: terminal	capability data base.	termcap(4)
terinfo: terminal	capability data base.	terinfo(4)
disks. dsk: winchester,	cartridge, and floppy	dsk(6)
(variant of ex for	casual users). /editor	edit(1)

files.	cat: concatenate and print	cat(1)
beautifier.	cb: C program	cb(1)
	cc: C compiler.	cc(1)
directory.	cd: change working	cd(1)
commentary of an SCCS delta.	cdc: change the delta	cdc(1)
ceiling, remainder,/ floor,	ceil, fmod, fabs: floor,	floor(3)
/ceil, fmod, fabs: floor,	ceiling, remainder, absolute/	floor(3)
BTOS. mkboot: reformat	CENTIX kernel and copy it to	mkboot(1)
uname: CENTIX system to	CENTIX system copy.	uucp(1)
uucp, uulog, uname:	CENTIX system to CENTIX/	uucp(1)
print name of current	CENTIX system. uname:	uname(1)
get name of current	CENTIX system. uname:	uname(2)
command execution. uux:	CENTIX-to-CENTIX system	uux(1)
uuto, uupick: public	CENTIX-to-CENTIX system file/	uuto(1)
flowgraph.	cflow: generate C	cflow(1)
delta: make a delta	(change) to an SCCS file.	delta(1)
of running process by	changing nice. /priority	renice(1)
pipe: create an interprocess	channel.	pipe(2)
terminal's local RS-232	channels. tp: controlling	tp(6)
stream. ungetc: push	character back into input	ungetc(3)

Index-10

user. cuserid: get	character login name of the	cuserid(3)
getchar, fgetc, getw: get	character or word from a/	getc(3)
/putchar, fputc, putw: put	character or word on a stream.	putc(3)
ascii: map of ASCII	character set.	ascii(5)
__tolower, toascii: translate	characters. /__toupper,	conv(3)
isctrl, isascii: classify	characters. /isprint, isgraph,	ctype(3)
tr: translate	characters.	tr(1)
directory.	chdir: change working	chdir(2)
/dfsck: file system consistency	check and interactive repair.	fsck(1)
lint: a C program	checker.	lint(1)
grpck: password/group file	checkers. pwck,	pwck(1)
copy file systems with label	checking. volcopy, labelit:	volcopy(1)
systems processed by fsck.	checklist: list of file	checklist(4)
file. sum: print	checksum and block count of a	sum(1)
chown,	chgrp: change owner or group	chown(1)
times: get process and	child process times.	times(2)
terminate. wait: wait for	child process to stop or	wait(2)
	chmod: change mode.	chmod(1)
file.	chmod: change mode of file.	chmod(2)

of a file.	chown: change owner and group	chown(2)
group.	chown, chgrp: change owner or	chown(1)
directory.	chroot: change root	chroot(2)
for a command.	chroot: change root directory	chroot(1)
isgraph, iscntrl, isascii:	classify characters. /isprint,	ctype(3)
uuclean: uucp spool directory	clean-up.	uuclean(1)
screen.	clear: clear terminal	clear(1)
clri:	clear i-node.	clri(1)
clear:	clear terminal screen.	clear(1)
status./ ferror, feof, exRespond: send a message to a	clearerr, fileno: stream client.	ferror(3) exrespond(2)
set a process alarm	clock. alarm:	alarm(2)
cron:	clock daemon.	cron(1)
used.	clock: report CPU time used.	clock(3)
ldclose, ldaclose:	close a common object file.	ldclose(3)
close:	close a file descriptor.	close(2)
descriptor.	close: close a file	close(2)
fclose, fflush:	close or flush a stream. clri: clear i-node.	fclose(3) clri(1)
appropriate a request	cmp: compare two files.	cmp(1)
line-feeds.	code. exServerRq:	exserverq(2)
deltas.	col: filter reverse	col(1)
comb:	comb: combine SCCS combine SCCS deltas.	comb(1) comb(1)

Index-12

common to two sorted files.	comm: select or reject lines	comm(1)
nice: run a	command at low priority.	nice(1)
change root directory for a	command. chroot:	chroot(1)
env: set environment for	command execution.	env(1)
uux: remote system	command execution.	uux(1)
quits. nohup: run a	command immune to hangups and	nohup(1)
interactive BTOS JCL. ofcli:	command line interpreter for	ofcli(1)
getyopt: parse	command options.	getopt(1)
locate executable file for	command. path:	path(1)
shell, the standard/ restricted	command programming language.	sh(1)
data and/ timex: time a	command; report process	timex(1)
system: issue a shell	command.	system(3)
test: condition evaluation	command.	test(1)
time: time a	command.	time(1)
argument list(s) and execute	command. xargs: construct	xargs(1)
intro: introduction to	commands and applicaton/	intro(1)
at, batch: execute	commands at a later/	at(1)
install: install	commands.	install(1)
cdc: change the delta	commentary of an SCCS delta.	cdc(1)
ar:	common archive file format.	ar(4)

editor output. a.out:	common assembler and link	a.out(4)
and archive files to	common formats. /object	convert(1)
routines. ldfcn:	common object file access	ldfcn(4)
ldopen, ldaopen: open a	common object file for/	ldopen(3)
/line number entries of a	common object file function.	ldread(3)
/ldaclose: close a	common object file.	ldclose(3)
read the file header of a	common object file. ldhread:	ldhread(3)
entries of a section of a	common object file. /number	ldseek(3)
file header of a	common object file. /seek to	ldohseek(3)
/entries of a section of a	common object file.	ldrseek(3)
/section header of a	common object file.	ldhread(3)
an indexed/name section of a	common object file. /seek to	ldsseek(3)
of a symbol table entry of a	common object file. /the index	idtbindex(3)
symbol table entry of a	common object file. /indexed	ldtbread(3)
seek to the symbol table of a	common object file. ldtbseek:	ldtbseek(3)
line number entries in a	common object file. linenum:	linenum(4)
nm: print name list of	common object file.	nm(1)
relocation information for a	common object file. reloc:	reloc(4)
scnhdr: section header for a	common object file.	scnhdr(4)

Index-14

line number information from a	common object file. /and	strip(1)
retrieve symbol name for	common object file symbol/	ldgetname(3)
table format. syms:	common object file symbol	syms(4)
filehdr: file header for	common object files.	filehdr(4)
ld: link editor for	common object files.	ld(1)
size: print section sizes of	common object files.	size(1)
comm: select or reject lines	common to two sorted files.	comm(1)
ipcs: report inter-process	communication facilities/	ipcs(1)
stdipc: standard interprocess	communication package (ftok).	stdipc(3)
diff: differential file	comparator.	diff(1)
cmp:	compare two files.	cmp(1)
SCCS file. sccsdiff:	compare two versions of an	sccsdiff(1)
diff3: 3-way differential file	comparison.	diff3(1)
dircmp: directory	comparison.	dircmp(1)
expression. regcmp, regex:	compile and execute regular	regcmp(3)
regexp: regular expression	compile and match routines.	regexp(5)
regcmp: regular expression	compile.	regcmp(1)
term: format of	compiled term file.	term(4)
cc: C	compiler.	cc(1)
tic: terminfo	compiler.	tic(1)
yacc: yet another	compiler-compiler.	yacc(1)

modest-sized/ bs: a	compiler/interpreter for	bs(1)
erf, erfc: error function and	complementary error function.	erf(3)
wait: await	completion of process.	wait(1)
pack, pcat, unpack:	compress and expand files.	pack(1)
table entry of a/ ldtbindex:	compute the index of a symbol	ldtbindex(3)
cu: call another	computer system.	cu(1)
cat:	concatenate and print files.	cat(1)
test:	condition evaluation command.	test(1)
system. lpadmin:	configure the LP spooling	lpadmin(1)
fwtmp, wtmpfix: manipulate	connect accounting records.	fwtmp(1)
an out-going terminal line	connection. dial: establish	dial(3)
brc, bcheckrc, rc, allrc,	conrc: system initialization/	brc(1)
fsck, dfsck: file system terminal.	consistency check and/ console: console	fsck(1) console(6)
Application Processor/ console:	console: control console terminal.	console(1) console(6)
math: math functions and	constants.	math(5)
mkfs:	construct a file system.	mkfs(1)
execute command. xargs:	construct argument list(s) and	xargs(1)
ls: list	contents of directory.	ls(1)
csplit:	context split.	csplit(1)
Processor/ console:	control Application	console(1)

Index-16

ioctl:	control device.	ioctl(2)
fcntl: file	control.	fcntl(2)
init, icode, telinit: process	control initialization.	init(1)
msgctl: message	control operations.	msgctl(2)
semctl: semaphore	control operations	semctl(2)
shmctl: shared memory	control operations.	shmctl(2)
fcntl: file	control options.	fcntl(5)
uucp status inquiry and job	control. uustat:	uustat(1)
vc: version	control.	vc(1)
interface. tty:	controlling terminal	tty(6)
RS-232 channels. tp:	controlling terminal's local	tp(6)
terminals. term:	conventional names for	term(5)
units:	conversion program.	units(1)
dd:	convert and copy a file.	dd(1)
floating-point number. atof:	convert ASCII string to	atof(3)
integers and/ l3tol, ltol3: and base-64 ASCII/ a64l, l64a:	convert between 3-byte	l3tol(3)
and archive files to/ /gmtime, asctime, tzset:	convert between long integer	a64l(3)
to string. ecvt, fcvt, gcvt	convert: convert object	convert(1)
scanf, fscanf, sscanf:	convert date and time to/	ctime(3)
archive files/ convert:	convert floating-point numbner	ecvt(3)
strtod, atof:	convert formatted input.	scanf(3)
	convert object and	convert(1)
	convert string to/	strtod(3)

strtol, atol, atoi:	convert string to integer.	strtol(3)
dd: convert and	copy a file	dd(1)
bcopy: interactive block	copy	bcopy(1)
cpio:	copy file archives in and out.	cpio(1)
access time. dcopy:	copy file systems for optimal	dcopy(1)
checking,. volcopy, labelit:	copy file systems with label	volcopy(1)
reformat CENTIX kernel and	copy it to BTOS. mkboot:	mkboot(1)
cp, ln, mv:	copy, link or move files.	cp(1)
system, ofcopy:	copy to or from the BTOS file	ofcopy(1)
system to CENTIX system	copy. /uuname: CENTIX	uucp(1)
system-to-computer system file	copy. /uupick: public computer	uuto(1)
file.	core: format of core image	core(4)
core: format of	core image file.	core(4)
mem, kmem:	core memory.	mem(6)
atan2: trigonometric/ sin,	cos, tan, asin, acos, atan,	trig(3)
functions. sinh,	cosh, tanh: hyperbolic	sinh(3)
sum: print checksum and block	count of a file.	sum(1)
wc: word	count.	wc(1)
files.	cp, ln, mv: copy, link or move	cp(1)
cpio: format of	cpio archive.	cpio(4)
and out.	cpio: copy file archives in	cpio(1)
archive.	cpio: format of cpio	cpio(4)

Index-18

preprocessor.	cpp: the C language	cpp(1)
binary directories.	cpset: install object files in	cpset(1)
clock: report	CPU time used.	clock(3)
rewrite an existing one.	creat: create a new file or	creat(2)
file. tmpnam, tempnam:	create a name for a temporary	tmpnam(3)
an existing one. creat:	create a new file or rewrite	creat(2)
fork:	create a new process.	fork(2)
tmpfile:	create a temporary file.	tmpfile(3)
channel. pipe:	create an interprocess	pipe(2)
files. admin:	create and administer SCCS	admin(1)
(slice). crup:	create file system partition	crup(1)
umask: set and get file	creation mask.	umask(2)
file.	cron: clock daemon.	cron(1)
crontab__user	crontab__user crontab	crontab(1)
cxref: generate C program	crontab file.	crontab(1)
optimization package.	cross reference.	cxref(1)
curses:	CRT screen handling and	curses(3)
partition (slice).	crup: create file system	crup(1)
generate DES encryption.	crypt, setkey, encrypt:	crypt(3)
terminal.	csplit: context split.	csplit(1)
for terminal.	ct: spawn getty to a remote	ct(1)
asctime, tzset: convert date/	ctermid: generate file name	ctermid(3)
	ctime, localtime, gmtime,	ctime(3)

debugger.	ctrace: C program	ctrace(1)
system.	cu: call another computer	cu(1)
uname: get name of	current CENTIX system	uname(2)
uname: get name of	current CENTIX system	uname(2)
activity. sact: print	current SCCS file editing	sact(1)
slot in the utmp file of the	current user. /find the	ttyslot(3)
getcwd: get path-name of	current working directory.	getcwd(3)
and optimization package.	curses: CRT screen handling	curses(3)
name of the user.	cuserid: get character login	cuserid(3)
of each line of a file.	cut: cut out selected fields	cut(1)
each line of a file. cut:	cut out selected fields of	cut(1)
cross reference.	cxref: generate C program	cxref(1)
command; report process	data and system/ /time a	timex(1)
smart modem capability	data base. modemcap:	modemcap(5)
termcap: terminal capability	data base.	termcap(4)
terminfo: terminal capability	data base.	terminfo(4)
/sgetl: access long integer	data in a machine-independent	sputl(3)
lock process, text, or	data in memory. plock:	plock(2)
prof: display profile	data.	prof(1)
call, stat:	data are turned by stat system	stat(5)

Index-20

brk, sbrk: change	data segment space allocation.	brk(2)
types: primitive system	data types.	types(5)
join: relational	database operator.	join(1)
tput: query terminfo	database.	tput(1)
/asctime, tzset: convert	date and time to string.	ctime(3)
date: print and set the date.	date.	date(1)
	date: print and set the	date(1)
	dc: desk calculator.	dc(1)
optimal access time.	dcopy: copy file systems for	dcopy(1)
file.	dd: convert and copy a	dd(1)
adb: absolute	debugger.	adb(1)
ctrace: C program	debugger.	ctrace(1)
fsdb: file system	debugger.	fsdb(1)
sdb: symbolic	debugger.	sdb(1)
names. basename, dirname:	deliver portions of path	basename(1)
file. tail:	deliver the last part of a	tail(1)
delta commentary of an SCCS	delta. cdc: change the	cdc(1)
file. delta: make a	delta (change) to an SCCS	delta(1)
delta. cdc: change the	delta commentary of an SCCS	cdc(1)
rmel: remove a	delta from an SCCS file.	rmel(1)
to an SCCS file.	delta: make a delta (change)	delta(1)
comb: combine SCCS	deltas.	comb(1)
cron: clock	demon.	cron(1)

mesg: permit or	deny messages.	mesg(1)
close: close a file	descriptor.	close(2)
dup: duplicate an open file	descriptor.	dup(2)
/wmsetid: associate a file	descriptor with a window.	wmsetid(3)
dc:	desk calculator.	dc(1)
file. access:	determine accessibility of a	access(2)
file:	determine file type.	file(1)
for finite width output	device. /fold long lines	fold(1)
master: master	device information table.	master(4)
ioctl: control	device.	ioctl(2)
devnm:	device name.	devnm(1)
	devnm: device name.	devnm(1)
blocks.	df: report number of free disk	df(1)
check and interactive/ fsck,	dfck: file system consistency	fsck(1)
terminal line connection.	dial: establish an out-going	dial(3)
bdiff: big	diff.	bdiff(1)
comparator.	diff: differential file	diff(1)
comparison.	diff3: 3-way differential file	diff3(1)
sdiff: side-by-side	difference program.	sdiff(1)
diff:	differential file comparator.	diff(1)
diff3: 3-way	differential file comparison.	diff3(1)
in large files and/ pilf,	dio: performance improvement	pilf(5)
directories.	dir: format of	dir(4)

Index-22

comparison.	dircmp: directory	dircmp(1)
improvement in large files and	direct I/O. /dio: performance	pilff(5)
install object files in binary	directories. cpset:	cpset(1)
dir: format of	directories.	dir(4)
ofls: list BTOS files and	directories.	ofls(1)
rm, rmdir: remove files or	directories.	rm(1)
cd: change working	directory.	cd(1)
chdir: change working	directory.	chdir(2)
chroot: change root	directory.	chroot(2)
uuclean: uucp spool	directory clean-up.	uuclean(1)
dircmp:	directory comparison.	dircmp(1)
unlink: remove	directory entry.	unlink(2)
chroot: change root	directory for a command.	chroot(1)
/make a lost + found	directory for fsck.	mklost + found(1)
ofDIDir, ofReadDirSector: BTOS	directory functions. ofCrDir,	ofdir(3)
path-name of current working	directory. getcwd: get	getcwd(3)
ls: list contents of	directory.	ls(1)
mkdir: make a	directory.	mkdir(1)
mkdir: move a	directory.	mkdir(1)
pwd: working	directory name.	pwd(1)
ordinary file. mknod: make a	directory, or a special or	mknod(2)

path names. basename,	dirname: deliver portions of	basename(1)
printers. enable,	disable: enable/disable LP	enable(1)
acct: enable or	disable process accounting.	acct(2)
type, modes, speed, and line	discipline. /set terminal	getty(1)
sadp:	disk access profiler.	sadp(1)
df: report number of free	disk blocks.	df(1)
update: provide	disk synchronization.	update(1)
du: summarize	disk usage.	du(1)
cartridge, and floppy	disks. dsk: winchester,	dsk(6)
mount, umount: mount and	dismount file system.	mount(1)
vi: screen-oriented (visual)	display editor based on ex.	vi(1)
prof:	display profile data.	prof(1)
hypot: Euclidean	distance function.	hypot(3)
/lcong48: generate uniformly	distributed pseudo-random/	drand48(3)
whodo: who is	doing what.	whodo(1)
/atof: convert string to	double-precision number.	strtod(3)
tdl: RS-232 terminal	download.	tdl(1)
nrnd48, mrnd48, jrand48./	drand48, erand48, lrand48.	drand48(3)
cartridge, and floppy/	dsk: winchester,	dsk(6)
usage.	du: summarize disk	du(1)
an object file.	dump: dump selected parts of	dump(1)

Index-24

hd: hexadecimal and ascii file	dump.	hd(1)
od: octal	dump.	od(1)
object file. dump:	dump selected parts of an	dump(1)
descriptor.	dup: duplicate an open file	dup(2)
descriptor. dup:	duplicate an open file	dup(2)
echo:	echo arguments.	echo(1)
	echo: echo arguments.	echo(1)
floating-point number to/	ecvt, fcvt, gcvt: convert	ecvt(3)
	ed, red: text editor.	ed(1)
program. end, etext,	edata: last locations in	end(3)
ofed, ofvi:	edit BTOS files.	ofed(1)
ofed, ofvi:	edit BTOS files.	ofvi(1)
(variant of ex for/	edit: text editor	edit(1)
sact: print current SCCS file	editing activity.	sact(1)
/visual display	editor based on ex.	vi(1)
ed, red: text	editor.	ed(1)
ex: text	editor.	ex(1)
files. ld: link	editor for common object	ld(1)
common assembler and link	editor output. a.out:	a.out(4)
sed: stream	editor.	sed(1)
for casual/ edit: text	editor (variant of ex	edit(1)
/user, real group, and	effective group IDs.	getuid(2)
and/ /getegid: get read user,	effective user, real group,	getuid(2)
split FORTRAN, ratfor, or	efl files, fsplit:	fsplit(1)

for a pattern. grep,	egrep, fgrep: search a file	grep(1)
enable/disable LP printers.	enable, disable:	enable(1)
accounting. acct:	enable or disable process	acct(2)
enable, disable:	enable/disable LP printers	enable(1)
encryption, crypt, setkey,	encrypt: generate DES	crypt(3)
setkey, encrypt: generate DES	encryption. crypt,	crypt(3)
locations in program.	end, etext, edata: last	end(3)
getgrgid, getgrnam, setgrent,	endgrent, fgetgrent: get group/	getgrent(3)
getpwnuid, getpwnam, setpwent,	endpwent, fgetpwent: get/	getpwent(3)
utmp/ /pututline, setutent,	endutent, utmpname: access	getut(3)
nlist: get	entries from name list.	nlist(3)
file. linenum: line number	entries in a common object	linenum(4)
file/ /manipulate line number	entries of a common object	ldlread(3)
common/ /seek to line number	entries of a section of a	ldlseek(3)
/ldnrseek: seek to relocation	entries of a section of a/	ldrseek(3)
utmp,wtmp: utmp and wtmp	entry formats.	utmp(4)
fgetgrent: get group file	entry. /setgrent, endgrent,	getgrent(3)
fgetpwent: get password file	entry. /setpwent, endpwent,	getpwent(3)

Index-26

utmpname: access utmp file	entry. /setutent, endutent,	getut(3)
object file symbol table	entry. /symbol name for common	ldgetname(3)
/the index of a symbol table	entry of a common object file.	ldtbindex(3)
/read an indexed symbol table	entry of a common object file.	ldtbread(3)
putpwent: write password file	entry.	putpwent(3)
quAdd: add a new	entry to a BTOS queue.	quadd(3)
unlink: remove directory	entry.	unlink(2)
command execution.	env: set environment for	env(1)
	environ: user environment.	environ(5)
profile: setting up an	environment at login time.	profile(4)
environ: user	environment.	environ(5)
execution. env: set	environment for command	env(1)
getenv: return value for	environment name.	getenv(3)
putenv: change or add value to	environment.	putenv(3)
interface, and terminal	environment. /terminal	tset(1)
rand48, jrand48, drand48,	erand48, lrand48, nrand48,	drand48(3)
complementary error function.	erf, erfc: error function and	erf(3)
complementary error/ erf,	erfc: error function and	erf(3)
system error/ perror,	errno, sys__errlist, sys__nerr:	perror(3)

complementary/ erf, erfc:	error function and	erf(3)
function and complementary	error function. /erfc: error	erf(3)
sys_errlist, sys_nerr: system	error messages. /errno,	perror(3)
to system calls and	error numbers. /introduction	intro(2)
matherr:	error-handling function.	matherr(3)
hashcheck: find spelling	errors. /hashmake, spellin,	spell(1)
terminal line/ dial:	establish an out-going	dial(3)
setmnt:	establish mount table.	setmnt(1)
in program. end:	etext, edata: last locations	end(3)
hypot:	Euclidean distance function.	hypot(3)
expression. expr:	evaluate arguments as an	expr(1)
test: condition	evaluation command.	test(1)
/text editor (variant of	ex for casual users).	edit(1)
display editor based on	ex: text editor.	ex(1)
obtain/ exQueryDfltResp Exch,	ex. /screen-oriented (visual)	vi(1)
exWait, exCheck:	exAllocExch, exDeallocExch:	exchanges(2)
quReadNext, quReadKeyed:	examine an ICC message queue.	exwait(2)
wait for the response.	examine BTOS queue.	quread(3)
obtain and abandon	exCall: Send a request and	excall(2)
message queue. exWait,	exchanges. /exDeallocExch:	exchanges(2)
	exCheck: examine an ICC	exwait(2)

Index-28

a file. locking:	exclusive access to regions of	locking(2)
abandon/ /exAllocExch, execlp, execvp: execute a/	exDeallocExch: obtain and execl, execv, execl, execl, execl,	exchanges(2) exec(2)
execvp: execute/ execl, execv,	execl, execl, execlp,	exec(2)
execl, execv, execl, execl,	execlp, execvp: execute a/	exec(2)
path: locate	executable file for command.	path(1)
execl, execlp, execvp: specific Application/ spawn:	execute a file. /execl, execute a process on a	exec(2) spawn(1)
specific/ spawnlp, spawnvp:	execute a process on a	spawn(3)
construct argument list(s) and	execute command. xargs:	xargs(1)
regex: compile and set environment for command	execute regular/ regcmp, execution. env:	regcmp(3) env(1)
sleep: suspend	execution for an interval.	sleep(1)
sleep: suspend	execution for interval.	sleep(3)
monitor: prepare	execution profile.	monitor(3)
spawnsrv: service spawn	execution requests.	spawnsrv(1)
profil:	execution time profile.	profil(2)
uux: remote system command	execution.	uux(1)
execvp: execute a/ execl,	execv, execl, execl, execl, execlp,	exec(2)
execute/ execl, execv, execl,	execl, execlp, execvp:	exec(2)
/execv, execl, execl, execlp,	execvp: execute a file.	exec(2)

system calls. link, unlink:	exercise link and unlink	link(1)
a new file or rewrite an	existing one. creat: create	creat(2)
process.	exit, __exit: terminate	exit(2)
exit,	__exit: terminate process.	exit(2)
exponential, logarithm,/	exp, log, log10, pow, sqrt:	exp(3)
pcat, unpack: compress	expand files. pack,	pack(1)
and	exponential, logarithm,	exp(3)
exp, log, log10, pow, sqrt:	power,/	exp(3)
expression.	expr: evaluate arguments	expr(1)
	as an	
routines. regexp: regular	expression compile and	regexp(5)
	match	
regcmp: regular	expression compile.	regcmp(1)
expr: evaluate arguments	expression.	expr(1)
as an		
compile and execute	expression. regcmp, regex:	regcmp(3)
regular		
exAllocExch,	exQueryDfltRespExch,	exchanges(2)
exDeallocExch:/		
server.	exRequest: Send a	exrequest(2)
	message to a	
client.	exRespond: send a	exrespond(2)
	message to a	
exCnxSendOnDealloc:	exSendOnDealloc,	exfinal(2)
make/		
request code.	exServeRq: appropriate a	exserverq(2)
ICC message queue.	exWait, exCheck: examine	exwait(2)
	an	
remainder,/ floor, ceil,	fabs: floor, ceiling,	floor(3)
fmod,		
factor:	factor a number	factor(1)
	factor: factor a number.	factor(1)
true,	false: provide truth values.	true(1)

Index-30

data in a machine-independent	fashion.. /access long integer	sputl(3)
finc:	fast incremental backup.	finc(1)
/calloc, malloc, mallinfo:	fast main memory allocator.	malloc(3) (fast version)
abort: generate an IOT	fault.	abort(3)
a stream.	fclose, fflush: close or flush	fclose(3)
	fcntl: file control.	fcntl(2)
	fcntl: file control options.	fcntl(5)
floating-point number/ecvt,	fcvt, gcvt: convert	ecvt(3)
fopen, freopen,	fdopen: open a stream.	fopen(3)
status inquiries. ferror,	feof, clearerr, fileno: stream	ferror(3)
fileno: stream status/	ferror, feof, clearerr,	ferror(3)
statistics for a file system.	ff: list file names and	ff(1)
stream. fclose,	fflush: close or flush a	fclose(3)
word from a/ getc, getchar,	fgetc, getw: get character or	getc(3)
getgrnam, setgrent, endgrent,	fgetgrent: get group file/	getgrent(3)
/getpwnam, setpwent, endpwent,	fgetpwent: get password file/	getpwent(3)
stream. gets,	fgets: get a string from a	gets(3)
pattern. grep, egrep,	fgrep: search a file for a	grep(1)
times. utime: set	file access and modification	utime(2)
ldfcn: common object	file access routines.	ldfcn(4)
determine accessibility of a	file. access:	access(2)

tar: tape	file archiver.	tar(1)
cpio: copy	file archives in and out.	cpio(1)
pwck, grpck: password/group	file checkers.	pwck(1)
chmod: change mode of	file.	chmod(2)
change owner and group of a	file. chown:	chown(2)
diff: differential	file comparator.	diff(1)
diff3: 3-way differential	file comparison.	diff3(1)
fcntl:	file control.	fcntl(2)
fcntl:	file control options.	fcntl(5)
system-to-computer system	file copy. /public computer	uuto(1)
core: format of core image	file.	core(4)
umask: set and get	file creation mask.	umask(2)
crontab--user crontab	file.	crontab(1)
fields of each line of a	file. cut: cut out selected	cut(1)
dd: convert and copy a	file.	dd(1)
a delta (change) to an SCCS	file. delta: make	delta(1)
close: close a	file descriptor.	close(2)
dup: duplicate an open	file descriptor.	dup(2)
wmsetid, wmsetids: associate a	file descriptor with a window.	wmsetid(3)
hd: hexadecimal and ascii	file: determine file type.	file(1)
selected parts of an object	file dump.	hd(1)
	file. dump: dump	dump(1)

Index-32

sact: print current SCCS	file editing activity.	sact(1)
endgrent, fgetgrent: get group	file entry. /setgrent,	getgrent(3)
fgetpwent: get password	file entry. /endpwent,	getpwent(3)
utmpname: access utmp	file entry. /endutent,	getut(3)
putpwent; write password	file entry.	putpwent(3)
execlp, execvp: execute a	file. /execv, execl, execve,	exec(2)
grep, egrep, fgrep: search a	file for a pattern.	grep(1)
path: locate executable	file for command.	path(1)
ldopen: open a common object	file for reading. ldopen,	ldopen(3)
aliases: aliases	file for sendmail.	aliases(5)
ar: common archive	file format.	ar(4)
intro: introduction to entries of a common object	file formats.	intro(4)
get: get a version of an SCCS	file function. /line number	ldread(3)
group: group	file.	get(1)
files. filehdr:	file.	group(4)
file. ldfhread: read the	file header for common object	filehdr(4)
ldohseek: seek to the optional	file header of a common object	ldfhread(3)
split: split a	file header of a common object/	ldohseek(3)
issue: issue identification	file into pieces.	split(1)
	file.	issue(4)

of a member of an archive	file. /read the archive header	ldahread(3)
close a common object	file. ldclose, ldaclose:	ldclose(3)
file header of a common object	file. ldhread: read the	ldhread(3)
a section of a common object	file. /line number entries of	ldseek(3)
file header of a common object	file. /seek to the optional	ldohseek(3)
a section of a common object	file. /relocation entries of	ldrseek(3)
header of a common object	file. /indexed/named section	ldhread(3)
section of a common object	file. /to an indexed/named	ldsseek(3)
table entry of a common object	file. /the index of a symbol	ldtindex(3)
table entry of a common object	file. /read an indexed symbol	ldtbread(3)
table of a common object	file. /seek to the symbol	ldtbseek(3)
entries in a common object	file. linenum: line number	linenum(4)
link: link to a	file.	link(2)
access to regions of a	file. locking: exclusive	locking(2)
mknod: build special	file.	mknod(1)
or a special or ordinary	file. /make a directory,	mknod(2)
ctermid: generate	file name for terminal.	ctermid(3)
mktemp: make a unique	file name.	mktemp(3)
a file system. ff: list	file names and statistics for	ff(1)
change the format of a text	file. newform:	newform(1)

Index-34

name list of common object	file. nm: print	nm(1)
null: the null	file.	null(6)
/find the slot in the utmp	file of the current user.	ttyslot(3)
Input/output on a BTOS	file. ofRead, ofWrite:	ofread(3)
ofRename: rename a BTOS	file.	ofrename(3)
one. creat: create a new	file or rewrite an existing	creat(2)
passwd: password	file.	passwd(4)
or subsequent lines of one	file. /lines of several files	paste(1)
soft-copy terminals. pg:	file perusal filter for	pg(1)
/rewind, ftell: reposition a	file pointer in a stream.	fseek(3)
lseek: move read/write	file pointer.	lseek(2)
activity/ fpsar:	File Processor system	fpsar(1)
prs: print an SCCS	file.	prs(1)
read: read from	file.	read(2)
for a common object	file. /relocation information	reloc(4)
remove a delta from an SCCS	file. rmdel:	rmdel(1)
bfs: big	file scanner.	bfs(1)
two versions of an SCCS	file. sccsdiff: compare	sccsdiff(1)
sccsfile: format of SCCS	file.	sccsfile(4)
header for a common object	file. scnhdr: section	scnhdr(4)
ofSetFileStatus: BTOS	File Status. ofGetFileStatus,	ofstatus(3)

stat, fstat: get	file status.	stat(2)
from a common object	file. /line number information	strip(1)
checksum and block count of a	file. sum: print	sum(1)
swrite: synchronous write on a	file.	swrite(2)
/symbol name for common object	file symbol table entry.	ldgetname(3)
syms: common object	file symbol table format.	syms(4)
and interactive/ fsck, dfck:	file system consistency check	fsck(1)
fsdb:	file system debugger.	fsdb(1)
names and statistics for a	file system. ff: list file	ff(1)
fs: format of	file system.	fs(4)
mkfs: construct a	file system.	mkfs(1)
umount: mount and dismount	file system. mount,	mount(1)
mount: mount a	file system.	mount(2)
copy to or from the BTOS	file system. ofcopy:	ofcopy(1)
crup: create	file system partition (slice).	crup(1)
ustat: get	file system statistics.	ustat(2)
mnttab: mounted	file system table.	mnttab(4)
umount: unmount a	file system.	umount(2)
access time. dcopy: copy	file systems for optimal	dcopy(1)
fsck. checklist: list of	file systems processed by	checklist(4)
volcopy, labelit: copy	file systems with label/	volcopy(1)
deliver the last part of a	file. tail:	tail(1)

Index-36

term: format of compiled term	file.	term(4)
tmpfile: create a temporary	file.	tmpfile(3)
create a name for a temporary	file. tmpnam, tempnam:	tmpnam(3)
and modification times of a	file. touch: update access	touch(1)
ftw: walk a	file tree.	ftw(3)
file: determine	file type.	file(1)
undo a previous get of an SCCS	file. unget:	unget(1)
report repeated lines in a	file. uniq:	uniq(1)
val: validate SCCS	file.	val(1)
write: write on a	file.	write(2)
umask: set	file-creation mode mask.	umask(1)
common object files.	filehdr: file header for	filehdr(4)
ferror, feof, clearerr,	fileno: stream status/	ferror(3)
create and administer SCCS	files. admin:	admin(1)
/improvement in large	files and direct I/O.	piif(5)
ofls: list BTOS	files and directories.	ofls(1)
cat: concatenate and print	files.	cat(1)
cmp: compare two	files.	cmp(1)
lines common to two sorted	files. comm: select or reject	comm(1)
cp, ln, mv: copy, link or move	files.	cp(1)
file header for common object	files. filehdr:	filehdr(4)

find: find	files	find(1)
frec: recover	files from a backup tape.	frec(1)
format specification in text	files. fspec:	fspec(4)
cpset: install object	files in binary directories.	cpset(1)
intro: introduction to special	files.	intro(6)
link editor for common object	files. ld:	ld(1)
lockf: record locking on	files.	lockf(3)
ofDelete: Allocate BTOS	files. /ofChangeFileLength.	ofcreate(3)
ofed, ofvi: edit BTOS	files.	ofeditors(1)
ofCloseAllFiles: Access BTOS	files. /ofCloseFile,	ofopenfile(3)
rm, rmdir: remove	files or directories.	rm(1)
/merge same lines of several	files or subsequent lines of/	paste(1)
unpack: compress and expand	files. pack, pcat,	pack(1)
pr: print	files.	pr(1)
section sizes of common object	files. size: print	size(1)
sort: sort and/or merge	files.	sort(1)
/object and archive	files to common formats.	convert(1)
what: identify SCCS	files.	what(1)
terminals. pg: file perusal	filter for soft-copy	pg(1)
nl: line numbering	filter.	nl(1)
col:	filter reverse line-feeds.	col(1)
/exCnxSendOnDealloc: make	final requests.	exfinal(2)

	finc: fast incremental backup.	finc(1)
find:	find files.	find(1)
	find: find files.	find(1)
hyphen:	find hyphenated words.	hyphen(1)
ttyname, isatty:	find name of a terminal.	ttyname(3)
object library. lorder:	find ordering relation of an	lorder(1)
hashmake, spellin, hashcheck:	find spelling errors. spell,	spell(1)
of the current user. ttyslot:	find the slot in the utmp file	ttyslot(3)
fold: fold long lines for tee: pipe	finite width output device.	fold(1)
	fitting.	tee(1)
atof: convert ASCII string to	floating-point number.	atof(3)
ecvt, fcvt, gcvt: convert	floating-point number to/	ecvt(3)
/modf: manipulate parts of	floating-point numbers.	frexp(3)
floor, ceiling, remainder,/ floor, ceil, fmod, fabs:	floor, ceil, fmod, fabs:	floor(3)
/cartridge, and	floor, ceiling, remainder,/	floor(3)
cflow: generate C	floppy disks.	dsk(6)
fclose, fflush: close or remainder,/ floor, ceil,	flow graph.	cflow(1)
finite width output device.	flush a stream.	fclose(3)
width output device. fold:	fmod, fabs: floor, ceiling,	floor(3)
	fold: fold long lines for	fold(1)
	fold long lines for finite	fold(1)

stream.	fopen, freopen, fdopen: open a	fopen(3)
	fork: create a new process.	fork(2)
ar: common archive file	format.	ar(4)
newform: change the	format of a text file.	newform(1)
i-node:	format of an i-node.	inode(4)
term:	format of compiled term file.	term(4)
core:	format of core image file.	core(4)
cpio:	format of cpio archive.	cpio(4)
dir:	format of directories.	dir(4)
fs:	format of file system.	fs(4)
sccsfile:	format of SCCS file.	sccsfile(4)
files. fspec:	format specification in text	fspec(4)
object file symbol table	format. syms: common	syms(4)
archive files to common	formats. /object and	convert(1)
intro: introduction to file	formats.	intro(4)
wtmp: utmp and wtmp entry	formats. utmp,	utmp(4)
scanf, fscanf, sscanf: convert	formatted input.	scanf(3)
/vfprintf, vsprintf: print	formatted output of a varargs/	vprintf(3)
reporter. fpsar:	fp system activity	fpsar(1)
fprintf, sprintf: print	formatted output. printf,	printf(3)
system activity/	fpsar: File Processor	fpsar(1)
word on a/ putc, putchar,	fputc, putw: put character or	putc(3)

Index-40

stream. puts,	fputs: put a string on a	puts(3)
input/output.	fread, fwrite: binary	fread(3)
backup tape.	frec: recover files from a	frec(1)
df: report number of	free disk blocks.	df(1)
memory allocator. malloc,	free, realloc, calloc: main	malloc(3)
malloc, mallinfo:/	free, realloc, calloc,	malloc(3)
malloc,		
stream. fopen,	freopen, fdopen: open a	fopen(3)
parts of floating-point/	frexp, ldexp, modf:	frexp(3)
	manipulate	
frec: recover files	from a backup tape.	frec(1)
/and line number	from a common object	strip(1)
information	file.	
getw: get character or	from a stream. /fgetc,	getc(3)
word		
gets, fgets: get a string	from a stream.	gets(3)
rm del: remove a delta	from an SCCS file.	rm del(1)
getopt: get option letter	from argument vector.	getopt(3)
read: read	from file.	read(2)
ncheck: generate names	from i-numbers.	ncheck(1)
nlist: get entries	from name list.	nlist(3)
ofcopy: copy to or	from the BTOS file	ofcopy(1)
	system.	
getpw: get name	from UID.	getpw(3)
	fs: format of file system.	fs(4)
formatted input. scanf,	fscanf, sscanf: convert	scanf(3)
a lost + found directory for	fsck. mklost + found:	mklost + found(1)
	make	
of file systems processed	fsck. checklist: list	checklist(4)
by		

consistency check and/	fsck, dfsck: file system	fsck(1)
	fsdb: file system debugger.	fsdb(1)
reposition a file pointer in/	fseek, rewind, ftell:	fseek(3)
size.	fsize: calculate file	fsize(1)
text files.	fspec: format specification in	fspec(4)
or efl files.	fsplit: split fortran, ratfor,	fsplit(1)
stat,	fstat: get file status.	stat(2)
pointer in a/ fseek, rewind,	ftell: reposition a file	fseek(3)
communication package	(ftok). /standard interprocess	stdipc(3)
	ftw: walk a file tree.	ftw(3)
error/ erf, erfc: error	function and complementary	erf(3)
and complementary error	function. /error function	erf(3)
gamma: log gamma	function. .	gamma(3)
hypot: Euclidean distance	function.	hypot(3)
of a common object file	function. /line number entries	ldlread(3)
matherr: error-handling	function.	matherr(3)
prof: profile within a	function.	prof(5)
math: math	functions and constants.	math(5)
j0, j1, yn, y0, y1, yn: Bessel	functions.	bessel(3)
logarithm, power, square root	functions. /sqrt: exponential,	exp(3)
remainder, absolute value	functions. /floor, ceiling,	floor(3)
ocurse: optimized screen	functions.	ocurses(3)

Index-42

BTOS directory	functions. /ofReadDirSector:	ofdir(3)
sinh, cosh, tanh: hyperbolic	functions.	sinh(3)
atan, atan2: trigonometric	functions. /tan, asin, acos,	trig(3)
fread,	fwrite: binary input/output.	fread(3)
connect accounting records.	fwtmp, wtmpfix: manipulate	fwtmp(1)
gamma: log	gamma function. gamma: log gamma function.	gamma(3) gamma(3)
number to string. ecvt, fcvt,	gcvt: convert floating-point	ecvt(3)
abort:	generate an IOT fault	abort(3)
cflow:	generate C flow graph.	cflow(1)
reference. cxref:	generate C program cross	cxref(1)
terminal. ctermid:	generate file name for	ctermid(3)
crypt, setkey, encrypt:	generate DES encryption.	crypt(3)
ncheck:	generate names from i-numbers.	ncheck(1)
lexical tasks. lex:	generate programs for simple	lex(1)
/srand48, seed48, lcong48:	generate uniformly distributed/	drand48(3)
srand: simple random-number	generator. rand,	rand(3)
gets, fgets:	get a string from a stream.	gets(3)
get:	get a version of an SCCS file.	get(1)
ulimit:	get and set user limits.	ulimit(2)
the user. cuserid:	get character login name of	cuserid(3)

getc, getchar, fgetc, getw:	get character or word from a/	getc(3)
nlist:	get entries from name list.	nlist(3)
umask: set and	get file creation mask.	umask(2)
stat, fstat:	get file status.	stat(2)
ustat:	get file system statistics.	ustat(2)
file.	get: get a version of an SCCS	get(1)
setgrent, endgrent, fgetgrent:	get group file entry.	getgrent(3)
getlogin:	get login name.	getlogin(3)
logname:	get login name.	logname(1)
msgget:	get message queue.	msgget(2)
getpw:	get name from UID.	getpw(3)
system. uname:	get name of current CENTIX	uname(2)
unset: undo a previous argument vector. getopt:	get of an SCCS file.	unset(1)
setpwent, endpwent, fgetpwent:	get option letter from	getopt(3)
working directory. getcwd:	get password file entry.	getpwent(3)
times. times:	get path-name of current	getcwd(3)
and/ getpid, getpgrp, getppid:	get process and child process	times(2)
/geteuid, getgid, getegid:	get process, process group,	getpid(2)
semget:	get real user, effective user,/	getuid(2)
shmget:	get set of semaphores.	semget(2)
wmlayout:	get shared memory segment.	shmget(2)
	get terminal's window layout.	wmlayout(3)

Index-44

tty:	get the terminal's name.	tty(1)
time:	get time.	time(2)
wmgetid:	get window ID.	wmgetid(3)
get character or word from a/ character or work from/ getc,	getc, getchar, fgetc, getw: getc, getw: get	getc(3) getc(3)
current working directory.	getcwd: get path-name of	getcwd(3)
getuid, geteuid, getgid, environment name.	getegid: get real user,/ getenv: return value for	getuid(2) getenv(3)
real user, effective/ getuid,	geteuid, getgid, getegid: get	getuid(2)
user./ getuid, geteuid, setgrent, endgrent,/ endgrent,/ getgrent,	getgid, getegid: get real getgrent, getgrgid, getgrnam,	getuid(2) getgrent(3) getgrent(3)
getgrent, getgrgid,	getgrgid, getgrnam, setgrent,	getgrent(3)
argument vector.	getgrnam, setgrent, endgrent,/ getlogin: get login name.	getgrent(3) getlogin(3)
process group, and/ getpid,	getopt: get option letter from getopt: parse command options.	getopt(3) getopt(1)
process, process group, and/ group, and/ getpid, getpgrp,	getpass: read a password. getpgrp, getppid: get process, getpid, getpgrp, getppid: get getppid: get process, process getpw: get name from UID.	getpass(3) getpid(2) getpid(2) getpid(2) getpw(3)

setpwent, endpwent,/ getpwent, getpwuid, endpwent,/ a stream. and terminal settings used by modes, speed, and line/ ct: spawn settings used by getty. getegid: get real user,/ pututline, setutent,/ setutent, endutent,/ getutent, setutent,/ getutent, getutid, from a/ getc, getchar, fgetc, convert/ ctime, localtime, setjmp, longjmp: non-local sag: system activity plot: subroutines. plot: /for typesetting view file for a pattern.	getpwent, getpwuid, getpwnam, getpwnam, setpwent, endpwent,/ getpwuid, getpwnam, setpwent, gets, fgets: get a string from getty. gettydefs: speed getty: set terminal type, getty to a remote terminal. gettydefs: speed and terminal getuid, geteuid, getgid, getutent, getutid, getutline, getutid, getutline, pututline, getutline, pututline, getw: get character or word gmtime, asctime, tzset: goto. graph. graphics interface. graphics interface graphs and slides. grep, egrep, fgrep: search a	getpwent(3) getpwent(3) getpwent(3) gets(3) gettydefs(4) getty(1) ct(1) gettydefs(4) getuid(2) getut(3) getut(3) getut(3) getc(3) ctime(3) setjmp(3) sag(1) plot(4) plot(3) mv(5) grep(1)
---	---	--

Index-46

/user, effective user, real	group, and effective group/	getuid(2)
/getppid: get process, process	group, and parent process IDs.	getpid(2)
chown, chgrp: change owner or	group.	chown(1)
endgrent, fgetgrent: get group:	group file entry. /setgrent, group file.	getgrent(3)
	group.: group file.	group(4)
setpgrp: set process	group ID.	setpgrp(2)
id: print user and	group IDs and names.	id(1)
real group, and effective	group IDs. /effective user,	getuid(2)
setuid, setgid: set user and	group IDs.	setuid(2)
newgrp: log in to a new	group.	newgrp(1)
chown: change owner and	group of a file.	chown(2)
a signal to a process or a	group of processes. /send	kill(2)
update, and regenerate	groups of programs. /maintain,	make(1)
checkers. pwck,	grpck: password/group file	pwck(1)
ssignal,	gsignal: software signals.	ssignal(3)
terminal download. tdl,	gtdl, ptdl: RS-232	tdl(1)
processing. shutdown,	halt: terminate all	shutdown(1)
varargs:	handle variable argument list.	varargs(5)
package. curses: CRT screen	handling and optimization	curses(3)

nohup: run a command immune to	hangups and quits.	nohup(1)
hcreate, hdestroy: manage	hash search tables hsearch,	hsearch(3)
spell, hashmake, spellin, /encrypt: generate	hashcheck: find spelling/ hashing encryption.	spell(1) crypt(3)
hashcheck: find/ spell, search tables. hsearch,	hashmake, spellin, hcreate, hdestroy: manage hash	spell(1) hsearch(3)
dump.	hd: hexadecimal and ascii file	hd(1)
tables, hsearch, hcreate,	hdestroy: manage hash search	hsearch(3C)
file. scnhdr: section	header for a common object	scnhdr(4)
files. filehdr: file	header for common object	filehdr(4)
file. ldfhread: read the file	header of a common object	ldfhread(3)
/seek to the optional file	header of a common object/	ldohseek(3)
/read an indexed/named section	header of a common object/	ldhread(3)
ldahread: read the archive	header of a member of an/	ldahread(3)
help: ask for	help: ask for help.	help(1)
dump. hd:	help.	help(1)
manage hash search tables.	hexadecimal and ascii file	hd(1)
sinh, cosh, tanh:	hsearch, hcreate, hdestroy:	hsearch(3)
hyphen: find	hyperbolic functions.	sinh(3)
	hyphen: find hyphenated words.	hyphen(1)
	hyphenated words.	hyphen(1)

Index-48

function.	hypot: Euclidean distance	hypot(3)
exWait, exCheck: examine an	ICC message queue.	exwait(2)
processor. pstat:	ICC statistics for	pstat(1)
control initialization. init,	icode, telinit: process	init(1)
semaphore set or shared memory	id. /remove a message queue,	ipcrm(1)
and names.	id: print user and group IDs	id(1)
setpgrp: set process group	ID.	setpgrp(2)
wmgetid: get window	ID.	wmgetid(3)
issue: issue	identification file.	issue(4)
what:	identify SCCS files.	what(1)
id: print user and group	IDs and names.	id(1)
group, and parent process	IDs. /get process, process	getpid(2)
group, and effective group	IDs. /effective user, real	getuid(2)
setgid: set user and group	IDs. setuid,	setuid(2)
core: format of core	image file.	core(4)
crash: examine system	images.	crash(1)
nohup: run a command	immune to hangups and quits.	nohup(1)
direct/ pilf, dio: performance	improvement in large files and	pilf(5)
finc: fast	incremental backup.	finc(1)
tgoto, tputs: terminal	independent operations.	termcap(3)
for formatting a permuted	index. /the macro package	mptx(5)

of a/ ldtbindex: compute the	index of a symbol table entry	ldtbindex(3)
a common/ ldtbread: read an	indexed symbol table entry of	ldtbread(3)
ldshread, ldnsbread: read an	indexed/named section header/	ldshread(3)
ldsseek, ldnsseek: seek to an	indexed/named section of a/	ldsseek(3)
control initialization.	init, icode, telinit: process	init(1)
inittab: script for the	init process.	inittab(4)
tellinit: process control	initialization. init, icode,	init(1)
rc, allrc, conrc: system	initialization shell scripts.	brc(1)
process. popen, pclose:	initiate pipe to/from a	popen(3)
process.	inittab: script for the init	inittab(4)
clri: clear	i-node.	clri(1)
	inode: format of an i-node	inode(4)
inode: format of an	i-node.	inode(4)
convert formatted	input. /fscanf, sscanf:	scanf(3)
push character back into	input stream. ungetc:	ungetc(3)
fread, fwrite: binary	input/output.	fread(3)
ofRead, ofWrite:	Input/output on a BTOS file.	ofread(3)
stdio: standard buffered	input/output package.	stdio(3)
fileno: stream status	inquiries. /feof, clearerr,	ferror(3)
uustat: uucp status	inquiry and job control.	uustat(1)
install:	install commands.	install(1)
	install: install commands.	install(1)
directories. cpset:	install object files in binary	cpset(1)

Index-50

tset: set terminal, terminal	interface, and terminal/	tset(1)
abs: return	integer absolute value.	abs(3)
/164a: convert between long	integer and base-64 ASCII/	a64i(3)
sputl, sgetl: access long	integer data in a/	sputl(3)
atol, atoi: convert string to	integer, strtol,	strtol(3)
/tol3: convert between 3-byte	integers and long integers.	l3tol(3)
3-byte integers and long	integers. /convert between	l3tol(3)
bcopy:	interactive block copy.	bcopy(1)
command line interpreter for	interactive BTOS JCL. ofcli:	ofcli(1)
system consistency check and	interactive repair. /file	fsck(1)
mt:	interface for magnetic tape.	mt(6)
lp: parallel printer	interface.	lp(6)
plot: graphics	interface.	plot(4)
plot: graphics	interface subroutines.	plot(3)
termio: general terminal	interface.	termio(6)
tty: controlling terminal	interface.	tty(6)
BTOS JCL. ofcli: command line	interpreter for interactive	ofcli(1)
pipe: create an	interprocess channel.	pipe(2)
facilities/ ipc: report	inter-process communication	ipc(1)
package/ stdipc: standard	interprocess communication	stdipc(3)
suspend execution for an	interval., sleep:	sleep(1)

sleep: suspend executin for	interval.	sleep(3)
commands and application/	intro: introduction to	intro(1)
formats.	intro: introduction to file	intro(4)
miscellany.	intro: introduction to	intro(5)
files.	intro: introduction to special	intro(6)
subroutines and libraries.	intro: introduction to	intro(3)
calls and error numbers.	intro: introduction to system	intro(2)
applicaton programs. intro:	introduction to commands and	intro(1)
intro:	introduction to file formats.	intro(4)
intro:	introduction to miscellany.	intro(5)
intro:	introduction to special files.	intro(6)
and libraries. intro:	introduction to subroutines	intro(3)
and error numbers. intro:	introduction to system calls	intro(2)
ncheck: generate names from	i-numbers.	ncheck(1)
in large files and direct	I/O. /performance improvement	pilf(5)
	ioctl: control device.	ioctl(2)
abort: generate an	IOT fault.	abort(3)
semaphore set or shared/	ipcrm: remove a message queue,	ipcrm(1)
communication facilities/	ipcs: report inter-process	ipcs(1)
/islower, isdigit, isxdigit,	isalnum, isspace, ispunct./	ctype(3)
isdigit, isxdigit, isalnum,/	isalpha, isupper, islower,	ctype(3)

Index-52

/isprint, isgraph, iscntrl,	isascii: classify characters.	ctype(3)
terminal. ttyname,	isatty: find name of a	ttyname(3)
/ispunct, isprint, isgraph,	iscntrl, isascii: classify/	ctype(3)
isalpha, isupper, islower,	isdigit, isxdigit, isalnum,/	ctype(3)
isspace, ispunct, isprint,	isgraph, iscntrl, isascii:/	ctype(3)
isalnum,/ isalpha,	islower, isdigit, isxdigit,	ctype(3)
isupper,		
/isalnum, isspace,	isprint, isgraph, iscntrl,/	ctype(3)
ispunct,		
isxdigit, isalnum, isspace,	ispunct, isprint, isgraph,/	ctype(3)
/isdigit, isxdigit, isalnum,	isspace, ispunct, isprint,/	ctype(3)
system:	issue a shell command.	system(3)
issue:	issue identification file.	issue(4)
file.	issue: issue identification	issue(4)
isxdigit, isalnum,/	isupper, islower, isdigit,	ctype(3)
isalpha,		
/isupper, islower, isdigit,	isxdigit, isalnum, isspace,/	ctype(3)
news: print news	items.	news(1)
functions.	j0, j1, jn, y0, y1, yn: Bessel	bessel(3)
functions. j0	j1, jn, y0, y1, yn: Bessel	bessel(3)
for interactive BTOS	JCL. /command line	ofcli(1)
	interpreter	
functions. j0, j1,	jn, y0, y1, yn: Bessel	bessel(3)
operator.	join: relational database	join(1)
lrnd48, nrnd48,	jrnd48, srnd48,	drand48(3)
mrnd48,	seed48,/	
mkboot: reformat CENTIX	kernel and copy it to	mkboot(1)
	BTOS.	

killall:	kill all active processes.	killall(1)
process or a group of/ processes.	kill: send a signal to a kill: terminate a process.	kill(2) kill(1)
mem,	killall: kill all active	killall(1)
3-byte integers and long/ integer and base-64/ a64l,	kmem: core memory. l3tol, ltol3: convert between 164a: convert between long	mem(6) l3tol(3) a64l(3)
copy file systems with with label checking. volcopy, scanning and processing arbitrary-precision arithmetic	label checking. /labelit: labelit: copy file systems language. awk: pattern language. bc:	volcopy(1) volcopy(1) awk(1) bc(1)
cpp: the C command programming	language preprocessor. language. /standard/restricted	cpp(1) sh(1)
get terminal's window /jrand48, srand48, seed48,	layout. wmlayout: lcong48: generate uniformly/	wmlayout(3) drand48(3)
object files.	ld: link editor for common	ld(1)
object file. ldclose,	ldaclose: close a common	ldclose(3)
header of a member of an/ file for reading. ldopen,	ldahread: read the archive ldaopen: open a common object	ldahread(3) ldopen(3)
common object file.	ldclose, ldaclose: close a	ldclose(3)
of floating-point/frexp, access routines.	ldexp, modf: manipulate parts ldfcn: common object file	frexp(3) ldfcn(4)

Index-54

of a common object file.	ldfhread: read the file header	ldfhread(3)
name for common object file/	ldgetname: retrieve symbol	ldgetname(3)
line number entries/ ldlread,	ldlinit, ldllitem: manipulate	ldlread(3)
number/ ldlread, ldlinit,	ldllitem: manipulate line	ldlread(3)
manipulate line number/	ldlread, ldlinit, ldllitem:	ldlread(3)
to line number entries/	ldlseek, ldlnlseek: seek	ldlseek(3)
number entries of a section/	ldlseek, ldlnlseek: seek to line	ldlseek(3)
entries of a section/ ldrseek,	ldnrseek: seek to relocation	ldrseek(3)
indexed/named/ ldshread,	ldnshread: read an	ldshread(3)
indexed/named/ ldsseek,	ldnsseek: seek to an	ldsseek(3)
file header of a common/ object file for reading.	ldohseek: seek to the optional	ldohseek(3)
relocation entries of a/ indexed/named section of a/	ldopen, ldaopen: open a common	ldopen(3)
indexed/named section of a/	ldrseek, ldnrseek: seek to	ldrseek(3)
of a symbol table entry of a/	ldshread, ldnshread: read an	ldshread(3)
symbol table entry of a/ table of a common object/	ldsseek, ldnsseek: seek to an	ldsseek(3)
getopt: get option	ldtbindex: compute the index	ldtbindex(3)
	ldtbread: read an indexed	ldtbread(3)
	ldtbseek: seek to the symbol	ldtbseek(3)
	letter from argument vector.	getopt(3)

simple lexical tasks.	lex: generate programs for	lex(1)
generate programs for simple	lexical tasks. lex:	lex(1)
update. lsearch,	lfind: linear search and	lsearch(3)
to subroutines and	libraries. /introduction	intro(3)
relation for an object	library. /find ordering	lorder(1)
portable/ ar: archive and	library maintainer for	ar(1)
ulimit: get and set user	limits.	ulimit(2)
an out-going terminal	line connection. /establish	dial(3)
type, modes, speed, and	line discipline. /set terminal	getty(1)
interactive/ ofcli: command	line interpreter for	ofcli(1)
line: read one	line.	line(1)
common object file. linenum:	line number entries in a	linenum(4)
/ldlinit, ldlitem: manipulate	line number entries of a/	ldlread(3)
ldlseek, ldnlseek: seek to	line number entries of a/	ldlseek(3)
strip: strip symbol and	line number information from a/	strip(1)
nl:	line numbering filter.	nl(1)
out selected fields of each	line of a file. cut: cut	cut(1)
send/cancel requests to an LP	line printer, lp, cancel:	lp(1)
lpset: set parallel	line printer options.	lpset(1)
lpr:	line printer spooler.	lpr(1)
	line: read one line.	line(1)
lsearch, lfind:	linear search and update.	lsearch(3)

col: filter reverse	line-feeds.	col(1)
in a common object file	linenum: line number entries	linenum(4)
files. comm: select or reject	lines common to two sorted	comm(1)
device. fold: fold long	lines for finite width output	fold(1)
head: give first few	lines.	head(1)
uniq: report repeated	lines in a file.	uniq(1)
of several files or subsequent	lines of one file. /same lines	paste(1)
subsequent/ paste: merge same	lines of several files or	paste(1)
link, unlink: exercise	link and unlink system calls.	link(1)
files. ld:	link editor for common object	ld(1)
a.out: common assembler and	link editor output.	a.out(4)
	link: link to a file.	link(2)
cp, ln, mv: copy,	link or move files.	cp(1)
link:	link to a file.	link(2)
and unlink system calls.	link, unlink: exercise link	link(1)
	lint: a C program checker.	lint(1)
ls:	list contents of directory.	ls(1)
directories. ofls:	list BTOS files and	ofls(1)
for a file system. ff:	list file names and statistics	ff(1)
nlist: get entries from name	list.	nlist(3)
nm: print name	list of common object file. .	nm(1)
by fsck. checklist:	list of file systems processed	checklist(4)

handle variable argument	list. varargs:	varargs(5)
output of a varargs argument	list. /print formatted	vprintf(3)
xargs: construct argument	list(s) and execute command.	xargs(1)
files. cp,	ln, mv: copy, link or move	cp(1)
tzset: convert data/ctime,	localtime, gmtime, asctime,	ctime(3)
command. path:	locate executable file for	path(1)
end, etext, edata: last	locations in program.	end(3)
data in memory. plock:	lock process, text, or	plock(2)
files.	lockf: record locking on	lockf(3)
regions of a file.	locking: exclusive access to	locking(2)
lockf: record	locking on files.	lockf(3)
gamma:	log gamma function.	gamma(3)
newgrp:	log in to a new group.	newgrp(1)
exponential, logarithm,/exp,	log, log10, pow, sqrt:	exp(3)
logarithm, power,/ exp, log,	log10, pow, sqrt: exponential,	exp(3)
/log10, pow, sqrt: exponential,	logarithm, power, square root/	exp(3)
getlogin: get	login name.	getlogin(3)
logname: get	login name.	logname(1)
cuserid: get character	login name of the user.	cuserid(3)
logname: return	login name of user.	logname(3)
passwd: change	login password.	passwd(1)
	login: sign on.	login(1)
setting up an environment at	login time. profile:	profile(4)
	logname: get login name.	logname(1)

Index-58

user.	logname: return login name of	logname(3)
a64l, l64a: convert between	long integer and base-64 ASCII/	a64l(3)
sputl, sgetl: access between 3-byte integers and	long integer data in a/ long integers. /lto13: convert	sputl(3) l3tol(3)
output device. fold: fold setjmp,	long lines for finite width longjmp: non-local goto.	fold(1) setjmp(3)
for an object library.	lorder: find ordering relation	lorder(1)
mklost + found: make a	lost + found directory for fsck.	mklost + found(1)
nice: run a command at requests to an LP line/ send/cancel requests to an	low priority. lp, cancel: send/cancel LP line printer. lp, cancel:	nice(1) lp(1) lp(1)
interface.	lp: parallel printer	lp(6)
disable: enable/disable	LP printers. enable,	enable(1)
lpshut, lpmove: start/stop the	LP request scheduler and move/	lpsched(1)
accept, reject: allow/prevent	LP requests.	accept(1)
lpadmin: configure the	LP spooling system.	lpadmin(1)
lpstat: print	LP status information.	lpstat(1)
spooling system.	lpadmin: configure the LP	lpadmin(1)
request/ lpsched, lpshut,	lpmove: start/stop the LP	lpsched(1)
	lpr: line printer spooler.	lpr(1)
start/stop the LP request/	lpsched, lpshut, lpmove:	lpsched(1)

printer options.	lpset: set parallel line	lpset(1)
LP request scheduler/ lpsched, information.	lpshut, lpmove: start/stop the	lpsched(1)
jranda48,/ dranda48, eranda48,	lpstat: print LP status	lpstat(1)
directory.	lranda48, nrand48, mrand48,	dranda48(3)
and update.	ls: list contents of	ls(1)
pointer.	lsearch, lfind: linear search	lsearch(3)
integers and long/ l3tol,	lseek: move read/write file	lseek(2)
values:	l3tol3: convert between 3-byte	l3tol(3)
/access long integer data in a	m4: macro processor.	m4(1)
permuted index. mptx: the	machine-dependent values.	values(5)
documents. mm: the MM	machine-independent fashion.	sputl(3)
typesetting/ mv: a troff	macro package for formatting	mptx(5)
m4:	macro package for formatting	mm(5)
in this manual. man:	macro processor.	mv(5)
send mail to users or read	macros for formatting entries	man(5)
users or read mail.	mail. mail, rmail:	mail(1)
mail, rmail: send	mail, rmail: send mail to	mail(1)
malloc, free, realloc, calloc:	mail to users or read mail.	mail(1)
/mallopt, mallinfo: fast	main memory allocator.	malloc(3)
regenerate groups of/ make:	main memory allocator.	malloc(3) (fast version)
	maintain, update, and	make(1)

Index-60

ar: archive and library	maintainer for portable/	ar(1)
SCCS file. delta:	make a delta (change) to an	delta(1)
mkdir:	make a directory.	mkdir(1)
or ordinary file. mknod:	make a directory, or a special	mknod(2)
mktemp:	make a unique file name.	mktemp(3)
exCnxSendOnDeal loc:	make final requests.	exfinal(2)
regenerate groups of/	make: maintain, update, and	make(1)
banner:	make posters.	banner(1)
session. script:	make typescript of terminal	script(1)
realloc, calloc, malloc,	mallinfo: fast main memory/	malloc(3) (fast version)
main memory allocator.	malloc, free, realloc, calloc:	malloc(3)
malloc, mallinfo: fast main/	malloc, free, realloc, calloc,	malloc(3)
malloc, free, realloc, calloc,	malloc, mallinfo: fast main/	malloc(3) (fast version)
/tfind, tdelete, twalk:	manage binary search trees.	tsearch(3)
hsearch, hcreate, hdestroy:	manage hash search tables.	hsearch(3)
wmop: window	management operations.	wmop(3)
window: window	management primitives.	window(6)
wm: window	management. .	wm(1)
records. fwtmp, wtmpfix:	manipulate connect accounting	fwtmp(1)
of/ ldlread, ldlini, ldliitem:	manipulate line number entries	ldlread(3)
frexp, ldexp, modf:	manipulate parts of/	frexp(3)
ascii:	map of ASCII character set.	ascii(5)
umask: set file-creation mode	mask.	umask(1)

set and get file creation	mask. umask:	umask(2)
table. master:	master device information	master(4)
information table.	master: master device	master(4)
regular expression compile and	match routines. regexp:	regexp(5)
math:	math functions and constants.	math(5)
constants.	math: math functions and	math(5)
function.	matherr: error-handling	matherr(3)
processor type.	mc68k, pdp11, u3b, vax:	machid(1)
	mem, kmem: core memory.	mem(6)
memcpy, memset: memory/	memcpy, memchr, memcmp,	memory(3)
memset: memory/ memcpy,	memchr, memcmp, memcpy,	memory(3)
operations. memcpy, memchr,	memcmp, memcpy, memset: memory	memory(3)
memcpy, memchr, memcmp,	memcpy, memset: memory/	memory(3)
free, realloc, calloc: main	memory allocator. malloc,	malloc(3)
malloc, mallinfo: fast main	memory allocator. /calloc,	malloc(3) (fast version)
shmctl: shared	memory control operations.	shmctl(2)
queue, semaphore set or shared	memory id. /remove a message	ipcrm(1)
mem, kmem: core	memory.	mem(6)
memcmp, memcpy, memset:	memory operations. /memchr,	memory(3)
shmop: shared	memory operations.	shmop(2)
text, or data in	memory. /lock process,	plock(2)
shmget: get shared	memory segment.	shmget(2)

Index-62

/memchr, memcmp, memcpy,	memset: memory operations.	memory(3)
sort: sort and/or files or subsequent/ paste:	merge files. merge same lines of several	sort(1) paste(1)
msgctl:	msg: permit or deny messages. message control operations.	msg(1) msgctl(2)
msgop:	message operations.	msgop(2)
exCheck: examine an ICC	message queue. exWait,	exwait(2)
msgget: get	message queue.	msgget(2)
or shared/ ipcrm: remove a	message queue, semaphore set	ipcrm(1)
exRespond: send a	message to a client.	exrespond(2)
exRequest: Send a	message to a server.	exrequest(2)
msg: permit or deny	messages.	msg(1)
sys__nerr: system error	messages. /errno, sys__errlist,	perror(3)
and copy it to BTOS.	mkboot: reformat CENTIX kernel	mkboot(1)
lost + found directory for/	mkdir: make a directory.	mkdir(1)
special or ordinary file.	mkfs: construct a file system.	mkfs(1)
name:	mklost + found: make a	mklost + found(1)
table.	mknod: build special file.	mknod(1)
chmod: change	mknod: make a directory, or a	mknod(2)
	mktemp: make a unique file	mktemp(3)
	mnttab: mounted file system	mnttab(4)
	mode.	chmod(1)

umask: set file-creation	mode mask.	umask(1)
chmod: change	mode of file.	chmod(2)
modemcap: smart	modem capability data base.	modemcap(5)
capability data base.	modemcap: smart modem	modemcap(5)
getty: set terminal type,	modes, speed, and line/	getty(1)
/compiler/interpreter for	modem-sized programs.	bs(1)
floating-point/ frexp,	modf: manipulate parts of	frexp(3)
ldexp,	modification times of a file.	touch(1)
touch: update access and	modification times.	utime(2)
utime: set file access	monitor: prepare execution	monitor(3)
and	monitor uucp network.	uusub(1)
profile.	more, page: text perusal.	more(1)
uusub:	Motorola/Intel.	swapshort(3)
translate byte orders to	/swaplong:	
mount:	mount a file system.	mount(2)
system. mount, umount:	mount and dismount file	mount(1)
	mount: mount a file	mount(2)
	system.	
setmnt: establish	mount table.	setmnt(1)
dismount file system.	mount, umount: mount	mount(1)
	and	
mnttab:	mounted file system table..	mnttab(4)
mmdir:	move a directory.	mmdir(1)
cp, ln, mv: copy, link or	move files.	cp(1)
lseek:	move read/write file	lseek(2)
	pointer.	

Index-64

the LP request scheduler and	move request. /start/stop	lpsched(1)
formatting a permuted index.	mptx: the macro package for	mptx(5)
/erand48, lrand48, nrand48,	mrnd48, jrand48, srand48,/	drand48(3)
operations.	msgctl: message control	msgctl(2)
	msgget: get message queue.	msgget(2)
	msgop: message operations.	msgop(2)
tape.	mt: interface for magnetic	mt(6)
package for typesetting/	mv: a troff macro	mv(5)
cp, ln,	mv: copy, link or move files.	cp(1)
	mvdir: move a directory.	mvdir(1)
i-numbers.	ncheck: generate names from	ncheck(1)
uusub: monitor uucp	network.	uusub(1)
a text file.	newform: change the format of	newform(1)
	newgrp: log in to a new group.	newgrp(1)
news: print	news items.	news(1)
	news: print news items.	news(1)
process.	nice; change priority of a	nice(2)
process by changing	nice. /of running	renice(1)
priority.	nice: run a command at low	nice(1)
	nl: line numbering filter.	nl(1)
list.	nlist: get entries from name	nlist(3)
object file.	nm: print name list of common	nm(1)
hangups and quits.	nohup: run a command immune to	nohup(1)

setjmp, longjmp:	non-local goto.	setjmp(3)
drand48, erand48, lrand48,	nrand48, mrand48, jrand48, /	drand48(3)
null: the	null file.	null(6)
	null: the null file.	null(6)
nl: line	numbering filter.	nl(1)
to/ convert: convert	object and archive files	convert(1)
ldfcn: common	object file access routines.	ldfcn(4)
dump selected parts of an	object file. dump:	dump(1)
ldopen, ldaopen: open a common	object file for reading.	ldopen(3)
number entries of a common	object file function. /line	ldlread(3)
ldaclose: close a common	object file. ldclose,	ldclose(3)
the file header of a common	object file. ldff:read: read	ldffread(3)
of a section of a common	object file. /number entries	ldlseek(3)
file header of a common	object file. /to the optional	ldohseek(3)
of a section of a common	object file. /entries	ldrseek(3)
header of a common	object file. /section	ldshread(3)
section header of a common	object file. /indexed/named	ldsseek(3)
symbol table entry of a common	object file. /the index of a	ldtbindex(3)
symbol table entry of a common	object file. /read an indexed	ldtbread(3)
the symbol table of a common	object file. /seek to	ldtbseek(3)
number entries in a common	object file. linenum: line	linenum(4)

Index-66

nm: print name list of common	object file.	nm(1)
information for a common	object file. /relocation	reloc(4)
section header for a common	object file. scnhdr:	scnhdr(4)
information from a common	object file. /and line number	strip(1)
entry. /symbol name for common	object file symbol table	ldgetname(3)
format. syms: common	object file symbol table	syms(4)
file header for common	object files. filehdr:	filehdr(4)
directories. cpset: install	object files in binary	cpset(1)
ld: link editor for common	object files.	ld(1)
print section sizes of common	object files. size:	size(1)
find ordering relation for an	object library. lorder:	lorder(1)
/exAllocExch, exDeallocExch:	obtain and abandon exchanges.	exchanges(2)
od:	octal dump.	od(1)
functions.	ocurse: optimized screen	ocurses(3)
	od: octal dump.	od(1)
Allocate BTOS/ ofCreate,	ofChangeFileLength, ofDelete:	ofcreate(3)
interpreter for interactive/	ofcli: command line	ofcli(1)
ofOpenFile, ofCloseFile,	ofCloseAllFiles: Access BTOS/	ofopenfile(3)
Access BTOS/ ofOpenFile,	ofCloseFile, ofCloseAllFiles:	ofopenfile(3)
BTOS file system.	ofcopy: copy to or from the	ofcopy(1)

ofReadDirSector: BTOS/	ofCrDir, ofDIDir,	ofdir(3)
ofDelete: Allocate BTOS/	ofCreate, ofChangeFileLength,	ofcreate(3)
ofCreate, ofChangeFileLength,	ofDelete: Allocate BTOS files.	ofcreate(3)
directory functions. ofCrDir,	ofDIDir, ofReadDirSector: BTOS	ofdir(3)
	ofed, ofvi: edit BTOS files.	ofeditors(1)
ofSetFileStatus: BTOS File/	ofGetFileStatus,	ofstatus(3)
directories.	ofls: list BTOS files and	ofls(1)
ofCloseAllFiles: Access BTOS/	ofOpenFile, ofCloseFile,	ofopenfile(3)
on a BTOS file.	ofRead, ofWrite: Input/output	ofread(3)
directory/ ofCrDir, ofDIDir,	ofReadDirSector: BTOS	ofdir(3)
	ofRename: rename a BTOS file.	ofrename(3)
Status. ofGetFileStatus, ofed,	ofSetFileStatus: BTOS File	ofstatus(3)
BTOS file. ofRead, reading. ldopen, ldaopen:	ofvi: edit BTOS files.	ofeditors(1)
	ofWrite: Input/output on a	ofread(3)
	open a common object file for	ldopen(3)
fopen, freopen, fdopen:	open a stream.	fopen(3)
dup: duplicate an	open file descriptor.	dup(2)
open:	open for reading or writing.	open(2)
writing.	open: open for reading or	open(2)
profiler. prf:	operating system	prf(6)
prfdc, prfsnap, prfpr:	operating system/	profiler(1)
memcmp, memcpy, memset: memory	operations. memccpy, memchr,	memory(3)

Index-68

msgctl: message control	operations.	msgctl(2)
msgop: message	operations.	msgop(2)
semctl: semaphore control	operations.	semctl(2)
semop: semaphore	operations.	semop(2)
shmctl: shared memory control	operations.	shmctl(2)
shmop: shared memory	operations.	shmop(2)
strcspn, strtok: string	operations. /strpbrk, strspn,	string(3)
tputs: terminal independent	operations. /tgetstr, tgoto,	termcap(3)
wmop: window management	operations.	wmop(3)
join: relational database	operator.	join(1)
dcopy: copy file systems for	optimal access time.	dcopy(1)
CRT screen handling and	optimization package.	curses(3)
	curses:	
ocurse:	optimized screen functions.	ocurses(3)
vector., getopt: get	option letter from argument	getopt(3)
common/ ldohseek: seek to the	optional file header of a	ldohseek(3)
fcntl: file control	options.	fcntl(5)
stty: set the	options for a terminal.	stty(1)
getopt: parse command	options.	getopt(1)
set parallel line printer	options. lpset:	lpset(1)
object library. lorder: find	ordering relation for an	lorder(1)

a directory, or a special or	ordinary file. mknod : make	mknod(2)
dial : establish an	out-going terminal line/	dial(3)
assembler and link editor	output. a.out : common	a.out(4)
long lines for finite width	output device. fold : fold	fold(1)
/vsprintf : print formatted	output of a varargs	vsprintf(3)
	argument/	
sprintf : print formatted	output. printf , fprintf ,	printf(3)
chown : change	owner and group of a file	chown(2)
chown , chgrp : change	owner or group.	chown(1)
and expand files.	pack , pcat , unpack :	pack(1)
	compress	
handling and optimization	package. curses : CRT	curses(3)
	screen	
view / mv : a troff macro	package for typesetting	mv(5)
sadc : system activity	package. sa1 , sa2 ,	sa(1)
report		
standard buffered	package. stdio :	stdio(3)
input/output		
interprocess	package (ftok). /standard	stdipc(3)
communication		
more ,	page : text perusal.	more(1)
lpset : set	parallel line printer	lpset(1)
	options.	
lp :	parallel printer interface.	lp(6)
process , process group ,	parent process IDs. /get	getpid(2)
and		
getopt :	parse command options.	getopt(1)
crup : create file system	partition (slice).	crup(1)

Index-70

	passwd: change login password.	passwd(1)
	passwd: password file.	passwd(4)
/endpwent, fgetpwent: get	password file entry.	getpwent(3)
putpwent: write	password file entry,.	putpwent(3)
passwd:	password file.	passwd(4)
getpass: read a	password.	getpass(3)
passwd: change login	password.	passwd(1)
pwck, grpck:	password/group file checkers.	pwck(1)
several files or subsequent/ for command.	paste: merge same lines of	paste(1)
dirname: deliver portions of	path: locate executable file	path(1)
directory. getcwd: get	path names. basename,	basename(1)
fgrep: search a file for a processing language.	path-name of current working	getcwd(3)
awk:	pattern. grep, egrep,	grep(1)
signal.	pattern scanning and	awk(1)
expand files. pack,	pause: suspend process until	pause(2)
a process. popen,	pcat, unpack: compress and	pack(1)
type. mc68k,	pclose: initiate pipe to/from	popen(3)
large files and/ pilf, dio:	pdp11, u3b, vax: processor	machid(1)
msg:	performance improvement in	pilf(5)
format. acct:	permit or deny messages.	msg(1)
sys__nerr: system error/	per-process accounting file	acct(4)
	perror, errno, sys__errlist,	perror(3)

terminals. pg: file	perusal filter for soft-copy	pg(1)
more, page: text	perusal.	more(1)
soft-copy terminals.	pg: file perusal filter for	pg(1)
split: split a file into	pieces.	split(1)
improvement in large files/	pilf, dio: performance	pilf(5)
channel.	pipe: create an interprocess	pipe(2)
tee:	pipe fitting.	tee(1)
popen, pclose: initiate	pipe to/from a process.	popen(3)
text, or data in/	plock: lock process:	plock(2)
interface.	kplot: graphics	plot(4)
subroutines.	plot: graphics interface	plot(3)
ftell: reposition a file	pointer in a stream. /rewind,	fseek(3)
lseek: move read/write file	pointer.	lseek(2)
to/from a process.	popen, pclose: initiate pipe	popen(3)
and library maintainer for	portable archives. /archive	ar(1)
basename, dirname: deliver	portions of path names.	basename(1)
banner: make	posters.	banner(1)
logarithm, exp, log, log10,	pow, sqrt: exponential,	exp(3)
exp, log, log10,	pow, sqrt: exponential,/	exp(3)
/exponential, logarithm,	power, square root/	exp(3)
monitor:	pr: print files.	pr(1)
cpp: the C language	prepare execution profile.	monitor(3)
unset: undo a	preprocessor.	cpp(1)
	previous get of an SCCS file.	unset(1)

Index-72

profiler.	prf: operating system	prf(6)
prfld, prfstat,	prfdc, prfsnap, prfpr:/	profiler(1)
prfsnap, prfpr:/	prfld, prfstat, prfdc,	profiler(1)
/prfstat, prfdc, prfsnap,	prfpr: operating system/	profiler(1)
prfld, prfstat, prfdc,	prfsnap, prfpr:/	profiler(1)
prfpr: operating/ prfld,	prfstat, prfdc, prfsnap,	profiler(1)
types:	primitive system data types.	types(5)
window: window management	primitives.	window(6)
prs:	print an SCCS file.	prs(1)
date:	print and set the date.	date(1)
number. apnum:	print Application Processor	apnum(1)
cal:	print calendar.	cal(1)
of a file. sum:	print checksum and block count	sum(1)
editing activity. sact:	print current SCCS file	sact(1)
cat: concatenate and	print files.	cat(1)
pr:	print files.	pr(1)
vprintf, vfprintf, vsprintf:	print formatted output of a/	vprintf(3)
printf, fprintf, sprintf:	print formatted output.	printf(3)
lpstat:	print LP status information.	lpstat(1)
object file. nm:	print name list of common	nm(1)
uname:	print name of system.	uname(1)
news:	print news items.	news(1)
object files. size:	print section sizes of common	size(1)

names. id:	print user and group IDs and	id(1)
lp: parallel	printer interface.	lp(6)
requests to an LP line	printer. /cancel: send/cancel	lp(1)
lpset: set parallel line	printer options.	lpset(1)
lpr: line	printer spooler.	lpr(1)
disable: enable/disable LP	printers. enable,	enable(1)
print formatted output.	printf, fprintf, sprintf:	printf(3)
nice: run a command at low	priority.	nice(1)
nice: change	priority of a process.	nice(2)
process/ renice: alter	priority of running	renice(1)
acct: enable or disable	process accounting.	acct(2)
alarm: set a	process alarm clock.	alarm(2)
times. times: get	process and child process	times(2)-
/priority of running	process by changing/	renice(1)
init, icode, telinit:	process control/	init(1)
timex: time a command; report	process data and system/	timex(1)
exit, __exit: terminate	process.	exit(2)
fork: create a new	process.	fork(2)
/getpgrp, getppid: get process,	process group, and parent/	getpid(2)
setpgrp: set	process group ID.	setpgrp(2)
process group, and parent	process IDs. /get process,	getpid(2)
inittab: script for the init	process.	inittab(4)
kill: terminate a	process.	kill(1)

Index-74

nice: change priority of a process.	process.	nice(2)
Application/ spawn: execute a	process on a specific	spawn(1)
spawnlp, spawnvp: execute a	process on a specific/	spawn(3)
kill: send a signal to a process or a group of/	process. popen, pclose:	kill(2)
initiate pipe to/from a	process, process group, and/	popen(3)
getpid, getpgrp, getppid: get	process status.	getpid(2)
ps: report	process, text, or data	ps(1)
in memory. plock: lock	process times.	plock(2)
times: get process and child	process to stop or terminate.	times(2)
wait: wait for child	process until signal.	wait(2)
pause: suspend	process.	pause(2)
wait: await completion of	processed by fsck. checklist:	wait(1)
list of file systems	processes. /send a signal	checklist(4)
to a process or a group of	processes.	kill(2)
killall: kill all active	processing language.	killall(1)
awk: pattern scanning and	processing.	awk(1)
shutdown, halt: terminate all	processor.	shutdown(1)
m4: macro	Processor number.	m4(1)
apnum: print Application	Processor pseudoconsole. .	apnum(1)
console: control Application		console(1)

ICC statistics for	processor. pstat:	pstat(1)
on a specific Applicaton	Processor. /execute a process	spawn(1)
on a specific Application	Processor. /execute a process	spawn(3)
activity/ fpsar: File	Processor system	fpsar(1)
mc68k, pdp11, u3b, vax:	processor type.	machid(1)
	prof: display profile data.	prof(1)
function.	prof: profile within a	prof(5)
profile.	profil: execution time	profil(2)
prof: display	profile data.	prof(1)
monitor: prepare execution	profil,e.	monitor(3)
profil: execution time	profile.	profil(2)
environment at login time.	profile: setting up an	profile(4)
prof:	profile within a function.	prof(5)
prf: operating system	profiler.	prf(6)
prfpr: operating system	profiler. /prfsnap,	profiler(1)
sadp: disk access	profiler.	sadp(1)
standard/restricted command	programming language. /the	sh(1)
update:	provide disk synchronization.	update(1)
/pdp11, u3b, u3b5, vax	provide truth value/	machid(1)
true, false:	provide truth values.	true(1)
	prs: print an SCCS file.	prs(1)
	ps: report process staus.	ps(1)
control Application Processor	pseudoconsole. console:	console(1)

Index-76

/generate uniformly distributed	pseudo-random numbers.	drand48(3)
for processor.	pstat: ICC statistics	pstat(1)
download. tdl, gtdl,	ptdl: RS-232 terminal	tdl(1)
	ptrace; process trace.	ptrace(2)
	ptx: permuted index.	ptx(1)
stream. ungetc:	push character back into input	ungetc(3)
put character or word on a/	putc, putchar, fputc, putw:	putc(3)
character or word on a/ putc,	putchar, fputc, putw: put	putc(3)
environment.	putenv: change or add value to	putenv(3)
entry.	putpwent: write password file	putpwent(3)
stream.	puts, fputs: put a string on a	puts(3)
getutent, getutid, getutline,	pututline, setutent, endutent, /	getut(3)
a/ putc, putchar, fputc,	putw: put character or word on	putc(3)
file checkers.	pwck, grpck: password/group	pwck(1)
	pwd: working directory name.	pwd(1)
	qsort: quicker sort.	qsort(3)
BTOS queue.	quAdd: add a new entry to a	quadd(3)
tput:	query temrinfo database.	tput(1)
examine an ICC message	queue. exWait, exCheck:	exwait(2)
msgget: get message	queue.	msgget(2)
add a new entry to a BTOS	queue. quAdd:	quadd(3)

quReadKeyed: examine BTOS	queue. quReadNext,	quread(3)
quRemove: take back a BTOS	queue request.	quremove(3)
ipcrm: remove a message	queue, semaphore set or shared/	ipcrm(1)
qsort:	quicker sort.	qsort(3)
command immune to hangups and	quits. nohup: run a	nohup(1)
queue. quReadNext,	quReadKeyed: examine BTOS	quread(3)
examine BTOS queue.	quReadNext, quReadKeyed:	quread(3)
queue request.	quRemove: take back a BTOS	quremove(3)
random-number generator.	rand, srand: simple	rand(3)
rand, srand: simple	random-number generator.	rand(3)
fsplit: split fortran,	ratfor, or efl files.	fsplit(1)
initialization/brc, bcheckrc,	rc, allrc, conrc: system	brc(1)
getpass:	read a password.	getpass(3)
entry of a common/ldtbread:	read an indexed symbol table	ldtbread(3)
header/ldshread, ldshread:	read an indexed/named section	ldshread(3)
read:	read from file.	read(2)
rmail: send mail to users or	read mail. mail,	mail(1)
line:	read one line.	line(1)
	read: read from file.	read(2)
member of an/ldahread:	read the archive header of a	ldahread(3)
common object file.	read the file header of a	ldfhread(3)
ldfhread:		

Index-78

open a common object file for	reading. ldopen, ldaopen:	ldopen(3)
open: open for	reading or writing.	open(2)
lseek: move	read/write file pointer.	lseek(2)
allocator. malloc, free,	realloc, calloc: main memory	malloc(3)
mallinfo: fast/ malloc, free,	realloc, calloc, mallopt,	malloc(3) (fast version)
specify what to do upon	receipt of a signal. signal:	signal(2)
lockf:	record locking on files.	lockf(3)
manipulate connect accounting	records. fwtmp, wtmpfix:	fwtmp(1)
tape. frec:	recover files from a backup	frec(1)
ed,	red: text editor.	ed(1)
it to BTOS. mkboot:	reformat CENTIX kernel and copy	mkboot(1)
execute regular expression.	regcmp, regex: compile and	regcmp(3)
expression compile.	regcmp: regular	regcmp(1)
make: maintain, update, and	regenerate groups of programs.	make(1)
regular expression. regcmp,	regex: compile and execute	regcmp(3)
compile and match routines.	regexp: regular expression	regexp(5)
locking: exclusive access to	regions of a file.	locking(2)
match routines. regexp:	regular expression compile and	regexp(5)
regcmp:	regular expression compile.	regcmp(1)
regex: compile and execute	regular expression. regcmp,	regcmp(3)
requests. accept,	reject: allow/prevent LP	accept(1)

sorted files. comm: select or	reject lines common two	comm(1)
lorder: find ordering	relation for an object/	lorder(1)
join:	relational database operator.	join(1)
for a common object file.	reloc: relocation information	reloc(4)
ldrseek, ldnrseek: seek to common object file. reloc:	relocation entries of a/ relocation information for a	ldrseek(3) reloc(4)
/fmod, fabs: floor, ceiling,	remainder, absolute value/	floor(3)
calendar.	reminder service.	calendar(1)
ct: spawn getty to a file. rmdel:	remote terminal. remove a delta from an SCCS	ct(1) rmdel(1)
semaphore set or/ ipcrm:	remove a message queue,	ipcrm(1)
unlink:	remove directory entry.	unlink(2)
rm, rmdir:	remove files or directories.	rm(1)
ofRename:	rename a BTOS file.	ofrename(3)
of running process by/ check and interactive	renice: alter priority repair. /system consistency	renice(1) fsck(1)
uniq: report	repeated lines in a file.	uniq(1)
clock:	report CPU time used.	clock(3)
communication/ ipcs:	report inter-process	ipcs(1)
blocks. df:	report number of free disk	df(1)
sa2, sadc: system activity	report package. sa1,	sar(1)
timex: time a command;	report process data and system/	timex(1)

Index-80

ps:	report process status.	ps(1)
file. uniq:	report repeated lines in a	uniq(1)
system activity	reporter. /Processor	fpsar(1)
sar: system activity	reporter.	sar(1)
stream, fseek, rewind, ftell:	reposition a file pointer in a	fseek(3)
reponse. exCall: Send a	request and wait for the	excall(2)
exServeRq: appropriate a	request code.	exserverq(2)
take back a BTOS queue	request. quRemove:	quremove(3)
/lpmove: start/stop the LP	request scheduler and move/	lpsched(1)
reject: allow/prevent LP	requests. accept,	accept(1)
exCnxSendOnDeal	request. exSendOnDealloc,	exfinal(2)
loc: make final		
LP request scheduler and move	requests. /start/stop the	lpsched(1)
service spawn execution	requests. spawnsvr:	spawnsvr(1)
syslocal: special system	requests.	syslocal(2)
lp, cancel: send/cancel	requests to an LP line/.	lp(1)
a request and wait for the	response. exCall: Send	excall(2)
common object file/ ldgetname:	retrieve symbol name for	ldgetname(3)
abs:	return integer absolute value.	abs(3)
logname:	return login name of user.	logname(3)
name. getenv:	return value for environment	getenv(3)

stat: data	returned by stat system call.	stat(5)
col: filter	reverse line-feeds.	col(1)
file pointer in a/ fseek, creat: create a new file or directories.	rewind, ftell: reposition a rewrite an existing one.	fseek(3) creat(2)
read mail. mail, SCCS file.	rm, rmdir: remove files or rmail: send mail to users or rmdel: remove a delta from an	rm(1) mail(1) rmdel(1)
directories. rm, chroot: change chroot: change	rmdir: remove files or root directory. root directory for a command.	rm(1) chroot(2) chroot(1)
logarithm, power, square	root functions. /exponential,	exp(3)
common object file access	routines. ldfcn:	ldfcn(4)
expression compile and match	routines. regexp: regular	regexp(5)
controlling terminal's local	RS-232 channels. tp:	tp(6)
tdl:	rs232 terminal download.	tdl(1)
standard/restricted/ sh, nice:	rsh: shell, the run a command at low priority.	sh(1) nice(1)
hangups and quits. nohup:	run a command immune to	nohup(1)
/alter priority of activity report package. report package. sal, editing activity. package. sa1, sa2,	running process by/ sa1, sa2, sadc: system sa2, sadc: system activity sact: print current SCCS file sadc: system activity report	renice(1) sar(1) sar(1) sact(1) sar(1)

	sar: system activity reporter.	sar(1)
profiler.	sadp: disk access	sadp(1)
graph.	sag: system activity	sag(1)
reporter.	sar: system activity	sar(1)
space allocation. brk,	sbrk: change data segment	brk(2)
formatted input.	scanf, fscanf, sscanf: convert	scanf(3)
bfs: big file	scanner.	bfs(1)
language. awk: pattern	scanning and processing	awk(1)
the delta commentary of an	SCCS delta. cdc: change	cdc(1)
comb: combine	SCCS deltas.	comb(1)
make a delta (change) to an	SCCS file. delta:	delta(1)
sact: print current	SCCS file editing activity.	sact(1)
get: get a version of an	SCCS file.	get(1)
prs: print an	SCCS file.	prs(1)
rmdel: remove a delta from an	SCCS file.	rmdel(1)
compare two versions of an	SCCS file. sccsdiff:	sccsdiff(1)
sccsfile: format of	SCCS file.	sccsfile(4)
undo a previous get of an	SCCS file. unget:	unget(1)
val: validate	SCCS file.	val(1)
admin: create and administer	SCCS files.	admin(1)
what: identify	SCCS files.	what(1)
of an SCCS file.	sccsdiff: compare two versions	sccsdiff(1)

	scsfile: format of SCCS file.	scsfile(4)
/start/stop the LP request	scheduler and move requests.	lpsched(1)
common object file.	scnhdr: section header for a	scnhdr(4)
clear: clear terminal	screen.	clear(1)
ocurse: optimized	screen functions.	ocurse(3)
optimization/ curses: CRT	screen handling and	curses(3)
display editor based on/ vi:	screen-oriented (visual)	vi(1)
inittab:	script for the init process.	inittab(4)
terminal session.	script: make typescript of	script(1)
system initialization shell	scripts. /rc, allrc, conrc:	brc(1)
	sdb: symbolic debugger.	sdb(1)
program.	sdiff: side-by-side difference	sdiff(1)
grep, egrep, fgrep:	search a file for a pattern	grep(1)
bsearch: binary	search a sorted table.	bsearch(3)
lsearch, lfind: linear	search and update.	lsearch(3)
hcreate, hdestroy: manage hash	search tables. hsearch,	hsearch(3)
tdelete, twalk: manage binary	search trees. tsearch, tfind,	tsearch(3)
object file. scnhdr:	section header for a common	scnhdr(4)
object/ /read an indexed/named	section header of a common	ldshread(3)
/to line number entries of a	section of a common object/	ldlseek(3)
/to relocation entries of a	section of a common object/	ldrseek(3)
/seek to an indexed/named	section of a common object/	ldsseek(3)

Index-84

files. size: print	section sizes of common object	size(1)
	sed: stream editor.	sed(1)
/mrand48, jrand48, srand48,	seed48, lcong48: generate/	drand48(3)
section of/ ldsseek, ldnseek:	seek to an indexed/named	ldsseek(3)
a section/ ldlseek, ldnlseek:	seek to line number entries of	ldlseek(3)
a section/ ldrseek, ldnrseek:	seek to relocation entries of	ldrseek(3)
header of a common/ ldohseek:	seek to the optional file	ldohseek(3)
common object file. ldtbseek:	seek to the symbol table of a	ldtbseek(3)
shmget: get shared memory	segment.	shmget(2)
brk, sbrk: change data	segment space allocation.	brk(2)
to two sorted files. comm:	select or reject lines common	comm(1)
of a file. cut: cut out	selected fields of each line	cut(1)
file. dump: dump	selected parts of an object	dump(1)
semctl:	semaphore control operations.	semctl(2)
semop:	semaphore operations.	semop(2)
ipcrm: remove a message queue,	semaphore set or shared memory/	ipcrm(1)
semget: get set of operations.	semaphores.	semget(2)
	semctl: semaphore control	semctl(2)
	semget: get set of semaphores.	semget(2)
	semop: semaphore operations.	semop(2)
exRspnd:	send a message to a client. .	exrespond(2)

exRequest:	Send a message to a server.	exrequest(2)
the response. exCall:	Send a request and wait for	excall(2)
a group of processes. kill:	send a signal to a process or	kill(2)
mail. mail, rmail:	send mail to users or read	mail(1)
line printer. lp, cancel:	send/cancel requests to an LP	lp(1)
aliases file for	sendmail. aliases:	aliases(5)
exRequest: Send a message to a	server.	exrequest(2)
make typescript of terminal	session. script:	script(1)
buffering to a stream	setbuf, setvbuf: assign	setbuf(3)
IDs. setuid,	setgid: set user and group	setuid(2)
getgrent, getgrgid, getgrnam,	setgrent, endgrent, fgetgrent:/	getgrent(3)
goto.	setjmp, longjmp: non-local	setjmp(3)
encryption. crypt,	setkey, encrypt: generate DES	crypt(3)
	setmnt: establish mount table.	setmnt(1)
	setpgrp: set process group ID.	setpgrp(2)
getpwent, getpwuid, getpwnam,	getpwent, endpwent, fgetpwent:/	getpwent(3)
environment/ cprofile:	setting up a C shell	cprofile(4)
login time. profile:	setting up an environment at	profile(4)
gettydefs: speed and terminal	setting used by getty.	gettydefs(4)
group IDs.	setuid, setgid: set user and	setuid(2)

	setuname: set name of system.	setuname(1)
/getutid, getutline, pututline,	setutent, endutent, utmpname:/	getut(3)
stream. setbuf,	setvbuf: assign buffering to a	setbuf(3)
data in a/ sputl,	sgetl: access long integer	sputl(3)
standard/restricted command/	sh, rsh: shell, the	sh(1)
operations. shmctl:	shared memory control	shmctl(2)
queue, semaphore set or	shared memory id. /a message	ipcrm(1)
shmop:	shared memory operations.	shmop(2)
shmget: get	shared memory segment.	shmget(2)
system: issue a	shell command.	system(3)
cprofile: setting up a C	shell environment at/	cprofile(4)
conrc: system initialization	shell scripts. /rc, allrc,	brc(1)
command programming/ sh, rsh:	shell, the standard/restricted	sh(1)
operations.	shmctl: shared memory control	shmctl(2)
segment.	shmget: get shared memory	shmget(2)
operations.	shmop: shared memory	shmop(2)
processing.	shutdown, halt: terminate all	shutdown(1)
program. sdiff:	side-by-side difference	sdiff(1)
login:	sign on.	login(1)
pause: suspend process until	signal.	pause(2)
what to do upon receipt of a	signal. signal: specify	signal(2)
upon receipt of a signal.	signal: specify what to do	signal(2)

of processes. kill: send a	signal to a process or a group	kill(2)
ssignal, gsignal: software	signals.	ssignal(3)
lex: generate programs for	simple lexical tasks.	lex(1)
generator. rand, srand:	simple random-number	rand(3)
atan, atan2: trigonometric/ functions.	sin, cos, tan, asin, acos, sinh, cosh, tanh: hyperbolic	trig(3)
fsize: calculate file	size.	fsize(1)
common object files	size: print section sizes of	size(1)
size: print section	sizes of common object files.	size(1)
an interval.	sleep: suspend execution for	sleep(1)
interval.	sleep: suspend execution for	sleep(3)
create file system partition	(slice). crup:	crup(1)
the/ ttyslot: find the	slot in the utmp file of	mv(5)
current/ ttyslot: find the	slot in the utmp file of the	ttyslot(3)
base. modemcap:	smart modem capability data	modemcap(5)
pg: file perusal filter for	soft-copy terminals.	pg(1)
ssignal, gsignal:	software signals.	ssignal(3)
sort:	sort and/or merge files.	sort(1)
qsort: quicker	sort.	qsort(3)
	sort: sort and/or merge files.	sort(1)
tsort: topological	sort.	tsort(1)

Index-88

or reject lines common to two	sorted files. comm: select	comm(1)
bsearch: binary search a	sorted table.	bsearch(3)
brk, sbrk: change data segment	space allocation.	brk(2)
specific Application/	spawn: execute a process on a	spawn(1)
spawnsvr: service terminal. ct:	spawn execution requests.	spawnsvr(1)
process on a specific/	spawn getty to a remote	ct(1)
execution requests.	spawnlp, spawnvnp: execute a	spawn(3)
a specific/ spawnlp,	spawnsvr: service spawn	spawnsvr(1)
spawn: execute a process on a	spawnvnp: execute a process on	spawn(3)
execute a process on a	specific Application/	spawn(1)
fspec: format	specific Application/ /spawnvnp:	spawn(3)
receipt of a signal. signal:	specification in text files.	fspec(4)
/set terminal type, modes,	specify what to do upon	signal(2)
used by getty. gettydefs:	speed, and line discipline.	getty(1)
hashcheck: find spelling/	speed and terminal settings	gettydefs(4)
spelling/ spell, hashmake,	spell, hashmake, spellin,	spell(1)
spellin, hashcheck: find	spellin, hashcheck: find	spell(1)
split:	spelling errors. /hashmake,	spell(1)
csplit: context	split a file into pieces.	split(1)
efl files. fsplit:	split.	csplit(1)
	split fortran, ratfor, or	fsplit(1)

pieces.	split: split a file into	split(1)
uuclean: uucp	spool directory clean-up.	uclean(1)
lpr: line printer	spooler.	lpr(1)
lpadmin: configure the LP	spooling system.	lpadmin(1)
output, printf, fprintf,	sprintf: print formatted	printf(3)
integer data in a/	sputl, sgetl: access long	sputl(3)
power,/ exp, log, log10, pow,	sqrt: exponential, logarithm,	exp(3)
exponential, logarithm, power	square root functions. /sqrt:	exp(3)
generator, rand,	srand: simple random-number	rand(3)
nrnd48, mrand48, jrand48,	srand48, seed48, lcong48:/	drand48(3)
input. scanf, fscanf,	sscanf: convert formatted	scanf(3)
signals.	ssignal, gsignal: software	ssignal(3)
package. stdio:	standard buffered input/output	stdio(3)
communication package/ stdipc:	standard interprocess	stdipc(3)
sh, rsh: shell, the	standard/restricted command/	sh(1)
lpsched, lpshut, lpmove:	start/stop the LP request/	lpsched(1)
system call.	stat: data returned by stat	stat(5)
stat: data returned by	stat, fstat: get file status.	stat(2)
ff: list file names and	stat system call.	stat(5)
processor. pstat: ICC	statistics for a file system.	ff(1)
ustat: get file system	statistics for	pstat(1)
	statistics.	ustat(2)

Index-90

lpstat: print LP	status information.	lpstat(1)
feof, clearerr, fileno: stream	status inquiries. ferror,	ferror(3)
control. uustat: uucp	status inquiry and job	uustat(1)
communication facilities	status. /report inter-process	ipcs(1)
ofSetFileStatus: BTOS File	Status. ofGetFileStatus.	ofstatus(3)
ps: report process	status.	ps(1)
stat, fstat: get file	status.	stat(2)
input/output package.	stdio: standard buffered	stdio(3)
	stime: set time.	stime(2)
wait for child process to	stop or terminate. wait:	wait(2)
strncmp, strcpy, strncpy,/	strcat, strncat, strcmp,	string(3)
/strcpy, strncpy, strlen, strncpy,/ strcat, strncat, /strncat, strcmp, strncmp,	strchr, strchr, strpbrk,/	string(3)
/strchr, strpbrk, strspn,	strcmp, strncmp, strcpy, strcpy, strncpy, strlen,/	string(3)
sed:	strcpn, strtok: string/ stream editor.	string(3)
fflush: close or flush a	stream. fclose,	sed(1)
fopen, freopen, fdopen: open a	stream.	fclose(3)
reposition a file pointer in a	stream. fseek, rewind, ftell:	fopen(3)
get character or word from a	stream. fgetc, rewind, ftell:	fseek(3)
fgets: get a string from a	stream. /getchar, fgetc, getw:	getc(3)
	stream. gets,	gets(3)

put character or word on a	stream. /putchar, fputc, putw:	putc(3)
puts, fputs: put a string on a	stream.	puts(3)
setvbuf: assign buffering to a	stream. setbuf,	setbuf(3)
/feof, clearerr, fileno:	stream status inquiries.	ferror(3)
push character back into input	stream. ungetc:	ungetc(3)
long integer and base-64 ASCII	string. /l64a: convert between	a64l(3)
convert date and time to floating-point number to	string. /asctime, tzset:	ctime(3)
gets, fgets: get a	string. /fcvt, gcvt: convert string from a stream.	ecvt(3) gets(3)
puts, fputs: put a	string on a stream.	puts(3)
strspn, strcspn, strtok:	string operations. /strpbrk,	string(3)
number. strtod, atof: convert	string to double-precision	strtod(3)
number. atof: convert ASCII	string to floating-point	atof(3)
strtol, atol, atoi: convert	string to integer.	strtol(3)
line number information/ number/ strip:	strip: strip symbol and strip symbol and line	strip(1) strip(1)
strncmp, strcpy, strncpy, strcpy, strncpy,/ strcat, strcat, strncat, strcmp, strcmp, strncmp, strcpy,	strlen, strchr, strchr,/strncat, strcmp, strncmp, strncmp, strcpy, strncpy,/strncpy, strlen, strchr,/	string(3) string(3) string(3) string(3)

Index-92

/strlen, strchr, strchr,	strpbrk, strspn, strcspn,/	string(3)
strncpy, strlen, strchr,	strrchr, strpbrk, strspn,/	string(3)
strchr, strrchr, strpbrk,	strspn, strcspn, strtok:/	string(3)
to double-precision number.	strtod, atof: convert string	strtod(3)
/strpbrk, strspn, strcspn,	strtok: string operations.	string(3)
string to integer.	strtol, atol, atoi: convert	strtol(3)
terminal.	stty: set the options for a	stty(1)
another user.	su: become super-user or	su(1)
intro: introduction to	subroutines and libraries.	intro(3)
plot: graphics interface	subroutines.	plot(3)
/same lines of several files or	subsequent lines of one file.	paste(1)
count of a file.	sum: print checksum and block	sum(1)
du:	summarize disk usage.	du(1)
sync: update the	super block.	sync(1)
sync: update	super-block.	sync(2)
su: become	super-user or another user.	su(1)
interval. sleep:	suspend execution of an	sleep(1)
interval. sleep:	suspend execution for	sleep(3)
pause:	suspend process until signal.	pause(2)
	swab: swap bytes.	swab(3)
swab:	swap bytes.	swab(3)
orders to/ swapshort,	swaplong: translate byte	swapshort(3)
byte orders to/	swapshort, swaplong: translate	swapshort(3)

file.	swrite: synchronous write on a	swrite(2)
information from/ strip: strip	symbol and line number	strip(1)
file/ ldgetname: retrieve	symbol name for common object	ldgetname(3)
name for common object file	symbol table entry. /symbol	ldgetname(3)
object/ /compute the index of a	symbol table entry of a common	ldtbindex(3)
ldtbread: read an indexed	symbol table entry of a common/	ldtbread(3)
syms: common object file	symbol table format.	syms(4)
object/ ldtbseek: seek to the	symbol table of a common	ldtbseek(3)
sdb:	symbolic debugger.	sdb(1)
symbol table format.	syms: common object file	syms(4)
	sync: update super-block	sync(2)
	sync: update the super block. .	sync(1)
update: provide disk	synchronization.	update(1)
swrite:	synchronous write on a file. .	swrite(2)
error/ perror, errno,	sys__errlist, sys__nerr: system	perror(3)
requests.	syslocal: special system	syslocal(2)
perror, errno, sys__errlist,	sys__nerr: system error/	perror(3)
binary search a sorted	table. bsearch:	bsearch(3)
for common object file symbol	table entry. /symbol name	ldgetname(3)
/compute the index of a symbol	table entry of a common object/	ldtbindex(3)

Index-94

file. /read an indexed symbol	table entry of a common object	ldtbread(3)
common object file symbol	table format. syms:	syms(4)
master device information	table. master:	master(4)
mnttab: mounted file system	table.	mnttab(4)
ldtbseek: seek to the symbol	table of a common object file.	ldtbseek(3)
setmnt: establish mount	table.	setmnt(1)
hdestroy: manage hash search	tables. hsearch, hcreate,	hsearch(3)
tabs: set	tabs on a terminal.	tabs(1)
	tabs: set tabs on a terminal.	tabs(1)
expand, unexpand: expand	tabs to spaces, and vice/	expand(1)
a file.	tail: deliver the last part of	tail(1)
request. quRemove:	take back a BTOS queue	quremove(3)
trigonometric/ sin, cos,	tan, asin, acos, atan, atan2:	trig(3)
sinh, cosh,	tanh: hyperbolic functions.	sinh(3)
tar:	tape file archiver.	tar(1)
recover files from a backup	tape. frec:	frec(1)
mt: interface for magnetic	tape.	mt(6)
	tar: tape file archiver.	tar(1)
programs for simple lexical	tasks. lex: generate	lex(1)
search trees, tsearch, tfind	tdelete, twalk: manage binary	tsearch(3)
	tdl: rs232 terminal download.	tdl(1)

	tee: pipe fitting.	tee(1)
initialization. init, icode,	telinit: process control	init(1)
temporary file. tmpnam,	tmpnam: create a name for a	tmpnam(3)
tmpfile: create a	temporary file.	tmpfile(3)
tmpnam: create a name for a	temporary file. tmpnam,	tmpnam(3)
terminals.	term: conventional names for	term(5)
term: format of compiled file.	term file. .	term(4)
	term: format of compiled term	term(4)
data base.	termcap: terminal capability	termcap(4)
termcap:	terminal capability data base. .	termcap(4)
terminfo:	terminal capability data base.	terminfo(4)
console: console	terminal.	console(6)
ct: spawn getty to a remote	terminal.	ct(1)
generate file name for	terminal. ctermid:	ctermid(3)
tdl: rs232	terminal download.	tdl(1)
/terminal interface, and	terminal environment.	tset(1)
/tgetstr, tgoto, tputs:	terminal independent/	termcap(3)
terminal/ tset: set terminal,	terminal interface, and	tset(1)
termio: general	terminal interface.	termio(6)
tty: controlling	terminal interface.	tty(6)
dial: establish an out-going	terminal line connection.	dial(3)

Index-96

clear: clear	terminal screen.	clear(1)
script: make typescript of	terminal session.	script(1)
getty. gettydefs: speed and	terminal settings used by	gettydefs(4)
stty: set the options for a	terminal.	stty(1)
tabs: set tabs on a	terminal.	tabs(1)
and terminal/ tset: set	terminal, terminal interface,	tset(1)
tty: get the name of the	terminal.	tty(1)
isatty: find name of a	terminal. ttyname,	ttyname(3)
and line/ getty: set	terminal type, modes, speed,	getty(1)
vt: virtual	terminal.	vt(6)
channels. tp: controlling	terminal's local RS-232	tp(6)
perusal filter for soft-copy	terminals. pg: file	pg(1)
term: conventional names for	terminals.	term(5)
wmlayout: get	terminal's window layout.	wmlayout(3)
kill:	terminate a process. _____	kill(1)
shutdown, halt:	terminate all processing.	shutdown(1)
exit, __exit:	terminate process.	exit(2)
for child process to stop or	terminate. wait: wait	wait(2)
tic:	terminfo compiler.	tic(1)
tput: query	terminfo database.	tput(1)
tic;	terminfo compiler.	terminfo(4)
interface:	termio: general terminal	termio(6)
command.	test: condition evaluation	test(1)
ed, red:	text editor.	ed(1)

ex:	text editor.	ex(1)
ex for casual/ edit:	text editor (variant of	edit(1)
change the format of a	text file. newform:	newform(1)
fspec: format specification in	text files.	fspec(4)
plock: lock process	text, or data in memory.	plock(2)
more, page:	text persual.	more(1)
strings: extract the ASCII	text strings in a file.	strings(1)
binary search types. tsearch,	tfind, tdelete, twalk: manage	tsearch(3)
tgetstr, tgoto, tputs:/	tgetent, tgetnum, tgetflag,	termcap(3)
tputs:/ tgetent, tgetnum,	tgetflag, tgetstr, tgoto,	termcap(3)
tgoto, tputs:/ tgetent,	tgetnum, tgetflag, tgetstr,	termcap(3)
tgetent, tgetnum, tgetflag,	tgetstr, tgoto, tputs:/	termcap(3)
/tgetnum, tgetflag, tgetstr,	tgoto, tputs: terminal/	termcap(3)
data and system/ timex:	tic: terminfo compiler.	tic(1)
time:	time a command; report process	timex(1)
commands at a later environment at login	time a command.	time(1)
systems for optimal access	time. /batch: execute	at(1)
profil: execution	time. /up a C shell	cprofile(4)
up an environment at login	time. dcopy: copy file	dcopy(1)
	time: get time.	time(2)
	time profile..	profil(2)
	time. profile: setting	profile(4)

Index-98

stime: set	time.	stime(2)
	time: time a command.	time(1)
time: get	time.	time(2)
tzset: convert date and	time to string. /asctime,	ctime(3)
clock: report CPU	time used.	clock(3)
process times.	times: get process and child	times(2)
update access and	times of a file. touch:	touch(1)
modification		
get process and child	times. times:	times(2)
process		
file access and	times. utime: set	utime(2)
modification		
process data and	timex: time a command;	timex(1)
system/	report	
file.	tmpfile: create a	tmpfile(3)
	temporary	
for a temporary file.	tmpnam, tmpnam: create	tmpnam(3)
	a name	
/tolower, __toupper,	toascii: translate	conv(3)
__tolower,	characters.	
popen, pclose: initiate	to/from a process.	popen(3)
pipe		
toupper, tolower,	__tolower, toascii:	conv(3)
__toupper,	translate/	
toascii: translate/	tolower, __toupper,	conv(3)
toupper,	__tolower,	
tsort:	topological sort.	tsort(1)
modification times of a	touch: update access and	touch(1)
file.		
translate/ toupper,	__toupper, __tolower,	conv(3)
tolower,	toascii:	
__tolower, toascii:	toupper, tolower,	conv(3)
translate/	__toupper,	
local RS-232 channels.	tp: controlling terminal's	tp(6)

database.	tput: query terminfo	tput(1)
/tgetflag, tgetstr, tgoto,	tputs: terminal independent/	termcap(3)
	tr: translate characters.	tr(1)
ptrace: process	trace.	ptrace(2)
swapshort, swaplong:	translate byte orders to/	swapshort(3)
/__toupper, __tolower, toascii:	translate characters.	conv(3)
tr:	translate characters.	tr(1)
ftw: walk a file	tree.	ftw(3)
twalk: manage binary search	trees: /tfind, tdelete,	tsearch(3)
tan, asin, acos, atan, atan2:	trigonometric functions. /cos,	trig(3)
typesetting view/ mv: a values.	troff macro package for	mv(5)
/u3b, u3b5, vax: provide	true, false: provide truth	true(1)
true, false: provide	truth value about your/	machid(1)
twalk: manage binary search/	truth values.	true(1)
interface, and terminal/	tsearch, tfind, tdelete,	tsearch(3)
	tset: set terminal, terminal	tset(1)
interface.	tsort: topological sort.	tsort(1)
	tty: controlling terminal	tty(6)
	tty: get the terminal's name.	tty(1)
a terminal.	ttyname, isatty: find name of	ttyname(3)
utmp file of the current/	ttyslot: find the slot in the	ttyslot(3)
tsearch, tfind, tdelete,	twalk: manage binary search/	tsearch(3)

Index-100

file: determine file	type.	file(1)
pdp11, u3b, vax: processor	type. mc68k,	machid(1)
getty: set terminal	type, modes, speed, and line/	getty(1)
ttytype: list of terminal	types by terminal number.	ttytype(4)
types.	types: primitive system data	types(5)
types: primitive system data	types.	types(5)
session. script: make	typescript of terminal	script(1)
/troff macro package for	typesetting view graphs/	mv(5)
/localtime, gmtime, asctime,	tzset: convert date and time/	ctime(3)
truth/ mc68k, pdp11,	u3b, u3b5, vax: provide	machid(1)
mc68k, pdp11, u3b,	u3b5, vax: provide truth/	machid(1)
getpw: get name from	UID.	getpw(3)
limits.	ulimit: get and set user	ulimit(2)
creation mask.	umask: set and get file	umask(2)
mask.	umask: set file-creation mode	umask(1)
file system. mount,	umount: mount and dismount	mount(1)
	umount: unmount a file system.	umount(2)
CTIX system.	uname: get name of current	uname(2)
	uname: print name of system.	uname(1)
an SCCS file.	unset: undo a previous get of	unset(1)

spaces, and/ expand,	unexpand: expand tabs to	expand(1)
get of an SCCS file	unget: undo a previous	unget(1)
into input stream.	ungetc: push character back	ungetc(3)
/seed48, lcong48: generate	uniformly distributed/	drand48(3)
a file.	uniq: report repeated lines in	uniq(1)
mktemp: make a	unique file name.	mktemp(3)
	units: conversion program.	units(1)
unlink system calls. link, entry.	unlink: exercise link and	link(1)
unlink: exercise link and	unlink: remove directory	unlink(2)
umount:	unlink system calls. link,	link(1)
files. pack, pcat,	unmount a file system.	umount(2)
	unpack: compress and expand	pack(1)
times of a file. touch:	update access and modification	touch(1)
of programs. make: maintain,	update, and regenerate groups	make(1)
lfind: linear search and synchronization	update. lsearch,	lsearch(3)
sync:	update: provide disk	update(1)
sync:	update super-block.	sync(2)
du: summarize disk	update the super block.	sync(1)
id: print	usage.	du(1)
	user and group IDs and names.	id(1)
setuid, setgid: set	user and group IDs.	setuid(2)
crontab--	user crontab file.	crontab(1)
character login name of the	user. cuserid: get	cuserid(3)

Index-102

/getgid, getegid: get real	user, effective user, read/	getuid(2)
environ:	user environment.	environ(5)
ulimit: get and set	user limits.	ulimit(2)
logname: return login name of	user.	logname(3)
/get real user, effective	user, real group, and/	getuid(2)
become super-user or another	user. su:	su(1)
the utmp file of the current	user. /find the slot in	ttyslot(3)
write: write to another	user.	write(1)
of ex for casual	users). /editor (variant	edit(1)
mail, rmail: send mail to	users or read mail.	mail(1)
wall: write to all	users.	wall(1)
statistics.	ustat: get file system	ustat(2)
modification times.	utime: set file access and	utime(2)
utmp, wtmp:	utmp and wtmp entry formats.	utmp(4)
endutent, utmpname: access	utmp file entry. /setutent,	getut(3)
ttyslot: find the slot in the	utmp file of the current user.	ttyslot(3)
entry formats.	utmp, wtmp: utmp and wtmp	utmp(4)
/putline, setutent, endutent,	utmpname: access utmp file/	getut(3)
clean-up.	uuclean: uucp spool directory	uuclean(1)
uusub: monitor	uucp network.	uusub(1)
uuclean:	uucp spool directory clean-up.	uuclean(1)

control. uustat:	uucp status inquiry and job	uustat(1)
between computer systems.	uucp, uulog, uuname: copy data	uucp(1)
between computer/ uucp,	uulog, uuname: copy data	uucp(1)
computer/ uucp, uulog,	uuname: copy data between	uucp(1)
system-to-computer/ uuto,	uupick: public computer	uuto(1)
and job control.	uustat: uucp status inquiry	uustat(1)
	uusub: monitor uucp network.	uusub(1)
system-to-computer system/	uuto, uupick: public computer	uuto(1)
execution.	uux: remote system command	uux(1)
	val: validate SCCS file.	val(1)
val:	validate SCCS file.	val(1)
u3b5, vax: provide truth	value about your/ /u3b,	machid(1)
abs: return integer absolute	value.	abs(3)
getenv: return	value for environment name.	getenv(3)
ceiling, remainder, absolute	value functions. /fabs: floor,	floor(3)
putenv: change or add	value to environment.	putenv(3)
values.	values: machine-dependent	values(5)
true, false: provide truth	values.	true(1)
values: machine-dependent	values.	values(5)
/print formatted output of a	varargs argument list.	vprintf(3)
argument list.	varargs: handle variable	varargs(5)

Index-104

varargs: handle	variable argument list.	varargs(5)
edit: text editor	(variant of ex for/	edit(1)
mc68k, pdp11, u3b,	vax: processor type.	machid(1)
	vc: version control.	vc(1)
option letter from argument	vector. getopt: get	getopt(3)
assert:	verify program assertion.	assert(3)
vc:	version control.	vc(1)
get: get a	version of an SCCS file.	get(1)
sccsdiff: compare two	versions of an SCCS file.	sccsdiff(1)
formatted output of/vprintf,	vprintf, vsprintf: print	vprintf(3)
display editor based on ex.	vi: screen-oriented (visual)	vi(1)
/package for typesetting	view graphs and slides.	mv(5)
on ex. vi: screen-oriented	(visual) display editor based	vi(1)
systems with label checking.	volcopy, labelit: copy file	volcopy(1)
print formatted output of a/	vprintf, vfprintf, vsprintf:	vprintf(3)
output of/ vprintf, vfprintf,	vsprintf: print formatted	vprintf(3)
process.	wait: await completion of	wait(1)
or terminate. wait:	wait for child process to stop	wait(2)
exCall: Send a request and to stop or terminate.	wait for the response.	excall(2)
	wait: wait for child process	wait(2)
ftw:	walk a file tree.	ftw(3)
	wall: write to all users.	wall(1)

	wc: word count.	wc(1)
	what: identify SCCS files.	what(1)
signal. signal: specify	waht to do upon receipt of a	signal(2)
whodo:	who is doing what.	whodo(1)
who:	who is on the system.	who(1)
	who: who is on the system.	who(1)
	whodo: who is doing what.	whodo(1)
fold long lines for finite and floppy disks. dsk:	width output device. fold:	fold(1)
wmgetid: get	winchester, cartridge,	dsk(6)
wmlayout: get terminal's	window ID.	wmgetid(3)
wmop:	window layout.	wmlayout(3)
	window management operations.	wmop(3)
window:	window management primitives.	window(6)
wm:	window management.	wm(1)
primitives:	window: window management	window(6)
a file descriptor with a	window. /wmsetids: associate	wmsetid(3)
	wm: window management.	wm(1)
	wmgetid: get window ID.	wmgetid(3)
window layout.	wmlayout: get terminal's	wmlayout(3)
operations.	wmop: window management	wmop(3)
file descriptor with a/	wmsetid, wmsetids: associate a	wmsetid(3)
descriptor with a/ wmsetid,	wmsetids: associate a file	wmsetid(3)

Index-106

cd: change	working directory.	cd(1)
chdir: change	working directory.	chdir(2)
get path-name of current	working directory. getcwd:	getcwd(3)
pwd:	working directory name.	pwd(1)
swrite: synchronous	write on a file.	swrite(2)
write:	write on a file.	write(2)
putpwent:	write password file entry.	putpwent(3)
wall:	write to all users.	wall(1)
write:	write to another user.	write(1)
	write: write on a file.	write(2)
	write: write to another user.	write(1)
open: open for reading or	writing.	open(2)
utmp, wtmp: utmp and	wtmp entry formats.	utmp(4)
formats. utmp,	wtmp: utmp and wtmp	utmp(4)
	entry	
accounting records.	wtmpfix: manipulate	fwtmp(1)
fwtmp,	connect	
list(s) and execute	xargs: construct argument	xargs(1)
command.		
j0, j1, jn,	y0, y1, yn: Bessel	bessel(3)
	functions.	
j0, j1, jn, y0,	y1, yn: Bessel functions.	bessel(3)
compiler-compiler.	yacc: yet another	yacc(1)
j0, j1, jn, y0, y1,	yn: Bessel functions.	bessel(3)

Title: _____

Form Number: _____ Date: _____

Burroughs Corporation is interested in your comments and suggestions regarding this manual. We will use them to improve the quality of your Product Information.

Please check type of suggestion: Addition Deletion Revision
 Error

Comments: _____

Name _____

Title _____

Company _____

Address _____

Street

City

State

Zip

Telephone Number (_____)
Area Code

Title: _____

Form Number: _____ Date: _____

Burroughs Corporation is interested in your comments and suggestions regarding this manual. We will use them to improve the quality of your Product Information.

Please check type of suggestion: Addition Deletion Revision
 Error

Comments: _____

Name _____

Title _____

Company _____

Address _____

Street

City

State

Zip

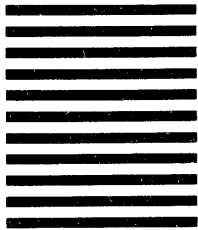
Telephone Number (_____)
Area Code



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 817 DETROIT, MI 48232

POSTAGE WILL BE PAID BY ADDRESSEE



Burroughs Corporation
Production Services – East
209 W. Lancaster Avenue
Paoli, Pa 19301 USA

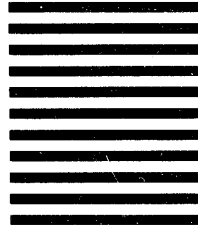
ATTN: Corporate Product Information



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 817 DETROIT, MI 48232

POSTAGE WILL BE PAID BY ADDRESSEE



Burroughs Corporation
Production Services – East
209 W. Lancaster Avenue
Paoli, Pa 19301 USA

ATTN: Corporate Product Information

