CLIX™

Programmer's and User's Reference Manual

INTERGRAPH

SE

CLIX™

Programmer's and User's Reference Manual

INTERGRAPH

DSYS18410

CLIX Programmer's & User's Reference Manual

January 1990

DSYS18412
Edition 1

Warranties and Liabilities

Trademarks

Classifications

This equipment is designed to comply with the requirements in Part 15 of the FCC rules for a class A computing device.

Copyrights

Additional References

The following UNIX System V documentation is required reference material. These documents can be purchased individually or in sets from Intergraph:

Title	**Release V.3**
AT&T UNIX System V User's Reference Manual	DSYS08110
AT&T UNIX System V User's Reference Addendum	DSYS19410
AT&T UNIX System V Administrator's Reference Manual	DSYS08310
AT&T UNIX System V Administrator's Reference Addendum	DSYS19710
AT&T UNIX System V Programmer's Reference Manual	DSYS08510
AT&T UNIX System V Programmer's Reference Addendum	DSYS19510

The following UNIX System V documentation is suggested reference material. The following documents can be purchased individually or in sets from Intergraph:

Title	**Release V.3**
AT&T UNIX System V User's Guide	DSYS08010
AT&T UNIX System V Programming Guide	DSYS08410
AT&T UNIX System V Administrator's Guide	DSYS08210

Ordering Information

To order any of these documents:

- Within the United States contact your Customer Engineer or Sales Account Representative.
- For International locations, contact the Intergraph subsidiary or distributor where you purchased your workstation.

Support Information

If you have trouble with the workstation/server or the procedures described in this guide, contact Intergraph Customer Support at 1–800–633–7248. International customers should contact the Intergraph subsidiary or distributor where the workstation was purchased.

Introduction

The *CLIX Programmer's & User's Reference Manual* describes the commands that constitute the basic software running on an Intergraph workstation or server, as well as system calls, library routines, file formats, and miscellaneous facilities used by programmers and users of a CLIX system running on an Intergraph workstation or server.

This manual supplements the AT&T UNIX System V documentation and thus includes only additions and changes found in the CLIX System.

The following documents provide related information:

- The *CLIX System Administrator's Reference Manual* describes the commands and special interfaces used by those who administer a CLIX system.
- The *CLIX System Guide* contains procedures and tutorials designed to give instructions in how to perform tasks and background information about when and why these tasks are desirable.

The *CLIX Programmer's & User's Reference Manual* is divided into the following sections:

(1) Commands

(2) System Calls

(2B) BSD System Calls

(2I) Intergraph System Calls

(3) Library Routines

(3C) and (3S) C Programming Language Utilities

(3B) BSD Library Routines

(3N) Intergraph Network Library Routines

(3R) RPC/XDR/YP Library Routines

(3A) Intergraph Synchronous/Asynchronous Library Routines

(4) File Formats

(5) Miscellaneous

The *CLIX System Administrator's Reference Manual* is divided into the following sections:

(1M) System Administrator Commands

(7) Special Interfaces

(7S) Special Files

(7B) BSD Network Interfaces

(7A) Asynchronous Interfaces

The *CLIX System Guide* is divided into the following sections:

Part 1: System Administrator's Tutorials

1. FFS Tutorial
2. FFS Check Tutorial
3. BSD LP Spooler Tutorial
4. NQS Tutorial
5. YP Tutorial

Part 2: System Administrator's Procedures

1. System Rebuild
2. New Product Delivery
3. System Reconfiguration
4. FFS Installation
5. BSD Network Configuration
6. NFS/YP Installation
7. NQS Installation

Part 3: Programmer's & User's Tutorials

1. Technical Programming Tutorial
2. PROC Debugging Tutorial
3. Network Programming Tutorial
4. BSD Porting Tutorial
5. Introductory Socket Tutorial
6. Advanced Socket Tutorial
7. NQS Tutorial
8. RCS Tutorial
9. RPC/XDR Tutorial

References

Throughout this manual, numbers following a command are intended for easy cross-reference.

- Look up references followed by (1), (2B), (2I), (3C), (3B), (3N), (3R), (3A), (4), or (5) in this document.
- Look up references followed by (1M), (7S), (7B), or (7A) in the in the *CLIX System Administrator's Reference Manual.*
- Look up all other references in the appropriate CLIX document.

If the references are not in the CLIX document, refer to the appropriate UNIX System V manual.

Format

Most sections begin with a page labeled *intro*. Entries following the *intro* page are arranged alphabetically and may consist of more than one page. Some entries describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "primary" name. (An example of such an entry is *chown*(1), which also describes the *chgrp* command.) To learn which manual page describes a secondary command, locate its name in the middle column of the "Permuted Index" and follow across that line to the name of the manual page listed in the right column.

All entries are based on a common format, but each part appears only where applicable:

- **NAME** gives the name(s) of the entry and briefly states its purpose.
- **SYNOPSIS** summarizes the use of the program being described. A few conventions are used, particularly in Section (1) (*Commands*):
    - **Boldface** strings are literals and are to be typed just as they appear.
    - *Italic* strings usually represent substitutable argument and program names found elsewhere in the manual.
    - Brackets [ ] around an argument indicate that the argument is optional.
    - Braces { } around arguments indicate that one of the arguments should be chosen.
    - Ellipses **...** are used to show that the previous argument may be repeated.
- **DESCRIPTION** provides an overview of the command.
- **EXAMPLES** gives examples of usage, where appropriate.
- **FILES** gives the file names that are built into the program.
- **SEE ALSO** offers pointers to related information.
- **DIAGNOSTICS** discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.
- **NOTES** gives information that may be helpful under the particular circumstances described.
- **WARNINGS** points out potential pitfalls.
- **BUGS** gives known bugs and sometimes deficiencies.
- **CAVEATS** gives details of the implementation that might affect usage.

- **IDENTIFICATION** gives the author of the program.

Table of Contents & Permuted Index

Preceding Section (1) is a "Table of Contents" (listing both primary and secondary command entries) and a "Permuted Index." Each line of the "Table of Contents" contains the name of a manual page (with secondary entries, if they exist) and an abstract of that page. Each line of the "Permuted Index" represents a permutation (or sorting) of a line from the "Table of Contents" into three columns. The lines are arranged so that a keyword or phrase begins the middle column. Use the "Permuted Index" by searching this middle column for a topic or command. When the desired entry has been found, the right column of that line lists the name of the manual page on which information corresponding to that keyword can be found. The left column contains the remainder of the permutation that began in the middle column.

Table of Contents

Table of Contents

5. Miscellaneous

Permuted Index

Permuted Index

	ab(1) Ada program beautifier	ab(1)
adb(1)	absolute debugger	adb(1)
	Ac(1) Ada compiler	ac(1)
accept(2B)	accept a connection on a socket	accept(2B)
sni_accept(3N)	accept a connection	sni_accept(3N)
a socket	accept(2B) accept a connection on	accept(2B)
.rhosts(4) remote user	access list	.rhosts(4)
search and print process	accounting files acctcom(1)	acctcom(1)
process accounting files	acctcom(1) search and print	acctcom(1)
readinfo(2I) read system	activity information	readinfo(2I)
CRM utility for monitoring I/O	activity topio(1)	topio(1)
CRM utility for monitoring system	activity topsys(1)	topsys(1)
Ac(1)	Ada compiler	ac(1)
alt(1)	Ada library tool	alt(1)
ab(1)	Ada program beautifier	ab(1)
adep(1)	Ada program makefile generator	adep(1)
	adb(1) absolute debugger	adb(1)
/inet_netof(3B) Internet	address manipulation routines	inet(3B)
netaddr(1) display network	address	netaddr(1)
generator	adep(1) Ada program makefile	adep(1)
aliases(4)	aliases file for sendmail(1M)	aliases(4)
rebuild the database for the mail	aliases file newaliases(1)	newaliases(1)
sendmail(1M)	aliases(4) aliases file for	aliases(4)
aligntrap(1) set/report	alignment trap disposition per/	aligntrap(1)
trap disposition per executable	aligntrap(1) set/report alignment	aligntrap(1)
fg_alloc(3A)	allocate a frame grabber	fg_alloc(3A)
fpe_coproc_alloc(3A)	allocate an FPE coprocessor	fpe_coproc_alloc(3A)
xio_allocef(3A) xio_deallocef(3A)	allocate/deallocate an event flag/	xio_allocef(3A)
check if a UCP priority is	already in use ucppri(2I)	ucppri(2I)
	alt(1) Ada library tool	alt(1)
ansitape(4)	ANSI standard magtape labels	ansitape(4)
program ansitape(1)	ANSI-standard magtape label	ansitape(1)
label program	ansitape(1) ANSI-standard magtape	ansitape(1)
labels	ansitape(4) ANSI standard magtape	ansitape(4)
link editor output	a.out(4) common assembler and	a.out(4)
introduction to commands and	application programs intro(1)	intro(1)
cpio(1) copy file	archives in and out	cpio(1)
vlock(2I) lock an	area of memory	vlock(2I)
vunlock(2I) unlock an	area of memory	vunlock(2I)
ftp(1)	ARPANET file transfer program	ftp(1)
	as(1) common assembler	as(1)
a.out(4) common	assembler and link editor output	a.out(4)
as(1) common	assembler	as(1)
existing optical disk/ odln(1)	associate another name with an	odln(1)
/return FORTRAN logical unit	associated with a file descriptor	fdtounit(3F)
unit /return the file descriptor	associated with a FORTRAN logical	fnum(3F)
/xio_wfland(3A) xio_wflor(3A)	asynchronous event control	xio_waitfr(3A)

sccstorcs(1)	build RCS file from SCCS file	sccstorcs(1)
/read data with error	byte from a serial port	aux_rawrd(3A)
values between host and network	byte order /ntohs(3B) convert	byteorder(3B)
bzero(3B) ffs(3B) bit and	byte string operations /bcmp(3B)	bstring(3B)
ntohl(3B) ntohs(3B) convert/	byteorder(3B) htonl(3B) htons(3B)	byteorder(3B)
bstring(3B) bcopy(3B) bcmp(3B)	bzero(3B) ffs(3B) bit and byte/	bstring(3B)
cc(1)	C compiler	cc(1)
or terminate FORTRAN I/O from	C /f77uninitio(3F) initialize	f77initio(3F)
data returned by stat system	call stat(5)	stat(5)
intro(2) introduction to system	calls and error numbers	intro(2)
CRM utility for monitoring system	calls and faults watcher(1)	watcher(1)
CSI port csi_ccan(3A)	cancel a specific command on a	csi_ccan(3A)
on a GPIB/ gpib_cancel(3A)	cancel all outstanding requests	gpib_cancel(3A)
serial port aux_cancel_modem(3A)	cancel modem change state on a	aux_cancel_modem(3A)
I/O on a CSI port csi_cancel(3A)	cancel outstanding asynchronous	csi_cancel(3A)
I/O on a PDI port pdi_cancel(3A)	cancel outstanding asynchronous	pdi_cancel(3A)
I/O on a SIF/ sif_cancel(3A)	cancel outstanding asynchronous	sif_cancel(3A)
I/O on an RLE/ rle_cancel(3A)	cancel outstanding asynchronous	rle_cancel(3A)
serial port aux_cancel(3A)	cancel outstanding read on a	aux_cancel(3A)
requests on a CSI/ csi_ucan(3A)	cancel unsolicited status	csi_ucan(3A)
coprocessor fpe_cancel_dma(3A)	cancel write request to an FPE	fpe_cancel_dma(3A)
printcap(4) BSD printer	capability database	printcap(4)
	cc(1) C compiler	cc(1)
/Intergraph software	certification documentation file	certnote.com(4)
software certification/	certnote.com(4) Intergraph	certnote.com(4)
disk files or/ odchown(1)	change file ownership of optical	odchown(1)
/aux_modem_nw(3A) get modem	change from a serial port	aux_modem(3A)
yppasswd(1)	change login password in YP	yppasswd(1)
chmod(1)	change mode	chmod(1)
chown(1) chgrp(1)	change owner or group	chown(1)
rcs(1)	change RCS file attributes	rcs(1)
aux_cancel_modem(3A) cancel modem	change state on a serial port	aux_cancel_modem(3A)
disk files or/ odchgrp(1)	change the file group of optical	odchgrp(1)
optical disk files or/ odchmod(1)	change the file protection of	odchmod(1)
kbmap(1)	change the keyboard layout	kbmap(1)
kbmap(2I)	change the keyboard layout	kbmap(2I)
rename(2B)	change the name of a file	rename(2B)
cnv_close(3A) close a CNV	channel	cnv_close(3A)
cnv_open(3A) open a CNV	channel	cnv_open(3A)
outstanding requests on a GPIB	channel /cancel all	gpib_cancel(3A)
gpib_close(3A) close a GPIB	channel	gpib_close(3A)
send commands to a GPIB	channel /gpib_cmd_nw(3A)	gpib_cmd(3A)
issue a local lockout to a GPIB	channel gpib_lockout(3A)	gpib_lockout(3A)
gpib_open(3A) open a GPIB	channel	gpib_open(3A)
perform a parallel poll of a GPIB	channel gpib_ppreq(3A)	gpib_ppreq(3A)
an IFC operation on a GPIB	channel gpib_reset(3A) conduct	gpib_reset(3A)
gpib_remote(3A) put a GPIB	channel in a remote state	gpib_remote(3A)
nlf_close(3A) close an NLF	channel	nlf_close(3A)
nlf_open(3A) open an NLF	channel	nlf_open(3A)

gpib_clear(3A) clear a GPIB	channel or device	gpib_clear(3A)
asynchronous I/O on an RLE	channel /cancel outstanding	rle_cancel(3A)
rle_close(3A) close an RLE	channel	rle_close(3A)
rle_open(3A) open an RLE	channel	rle_open(3A)
establish parameters for an RLE	channel rle_setup(3A)	rle_setup(3A)
asynchronous I/O on a SIF	channel /cancel outstanding	sif_cancel(3A)
sif_close(3A) close a SIF	channel	sif_close(3A)
sif_open(3A) open a SIF	channel	sif_open(3A)
establish parameters for a SIF	channel sif_setup(3A)	sif_setup(3A)
already in use ucppri(2I)	check if a UCP priority is	ucppri(2I)
ci(1)	check in RCS revisions	ci(1)
co(1)	check out RCS revisions	co(1)
chown(1)	chgrp(1) change owner or group	chown(1)
	chmod(1) change mode	chmod(1)
group	chown(1) chgrp(1) change owner or	chown(1)
	ci(1) check in RCS revisions	ci(1)
rcsclean(1)	clean up working files	rcsclean(1)
gpib_clear(3A)	clear a GPIB channel or device	gpib_clear(3A)
ucpclr(2I)	clear process UCP priority	ucpclr(2I)
clh(4) Intergraph network	clearinghouse database	clh(4)
clh(1) Intergraph network	clearinghouse management program	clh(1)
clearinghouse management program	clh(1) Intergraph network	clh(1)
clearinghouse database	clh(4) Intergraph network	clh(4)
object and property	clh_vbyop(3N) lookup value by	clh_vbyop(3N)
ypprot_err(3R) YP	client interfac /yperr_string(3R)	ypclnt(3R)
shell (command interpreter) with	C-like syntax csh(1) a	csh(1)
getcpuid(2I) return	CLIPPER processor identifier	getcpuid(2I)
processor type truth/ machid(1)	clipper(1) ns32000(1) vax(1) get	machid(1)
utd(1) copy between MS-DOS and	CLIX dtu(1)	dtu(1)
DNP remote login DECnet or	CLIX node. sethost(1)	sethost(1)
crm(1)	CLIX Resource Monitor	crm(1)
cnv_close(3A)	close a CNV channel	cnv_close(3A)
sni_close(3N)	close a connection	sni_close(3N)
csi_close(3A)	close a CSI port	csi_close(3A)
gpib_close(3A)	close a GPIB channel	gpib_close(3A)
pdi_close(3A)	close a PDI port	pdi_close(3A)
aux_close(3A)	close a serial port	aux_close(3A)
sif_close(3A)	close a SIF channel	sif_close(3A)
nlf_close(3A)	close an NLF channel	nlf_close(3A)
rle_close(3A)	close an RLE channel	rle_close(3A)
cnv_close(3A) close a	CNV channel	cnv_close(3A)
cnv_open(3A) open a	CNV channel	cnv_open(3A)
	cnv_close(3A) close a CNV channel	cnv_close(3A)
	cnv_open(3A) open a CNV channel	cnv_open(3A)
	co(1) check out RCS revisions	co(1)
syntax csh(1) a shell	(command interpreter) with C-like	csh(1)
netcp(1) DNP copy	command	netcp(1)
csi_ccan(3A) cancel a specific	command on a CSI port	csi_ccan(3A)
/execute the specified	command on remote system	fmu_rcmd(3N)

FORTRAN/ fnum(3F) return the file	descriptor associated with a	fnum(3F)
unit associated with a file	descriptor /FORTRAN logical	fdtounit(3F)
getdtablesize(2B) get	descriptor table size	getdtablesize(2B)
priority of the/ fg_priority(3A)	determine frame buffer output	fg_priority(3A)
window size fg_size(3A)	determine the frame grabber	fg_size(3A)
clear a GPIB channel or	device gpib_clear(3A)	gpib_clear(3A)
parallel poll response of a GPIB	device /configure the	gpib_ppconf(3A)
read data from a GPIB	device /gpib_read_nw(3A)	gpib_read(3A)
conduct a serial poll of a GPIB	device gpib_spreq(3A)	gpib_spreq(3A)
gpib_trigger(3A) trigger a GPIB	device	gpib_trigger(3A)
write data to a GPIB	device /gpib_write_nw(3A)	gpib_write(3A)
gpib_local(3A) return a GPIB	device to local control	gpib_local(3A)
/send a message to console	devices on the local XNS network	netmsg(1)
/unconfigure a GPIB	device's parallel poll response	gpib_ppuconf(3A)
qdev(1) display the status of NQS	devices	qdev(1)
ratfor(1) rational FORTRAN	dialect	ratfor(1)
group of optical disk files or	directories /change the file	odchgrp(1)
of optical disk files or	directories /the file protection	odchmod(1)
of optical disk files or	directories /file ownership	odchown(1)
list contents of optical disk	directories odls(1)	odls(1)
odmkdir(1) create optical disk	directories	odmkdir(1)
rename optical disk files or	directories odmv(1)	odmv(1)
odrmdir(1) delete optical disk	directories	odrmdir(1)
rm(1) rmdir(1) remove files or	directories	rm(1)
/recover(4) state(4) status daemon	directory and file formats	statmon(4)
/DNP command that lists the	directory contents on a remote/	netls(1)
dls(1) list contents of MS-DOS	directory	dls(1)
ls(1) list contents of	directory	ls(1)
ypfiles(4) the YP database and	directory structure	ypfiles(4)
odcd(1) set the current default	directory used by optical disk/	odcd(1)
/display the current default	directory used by optical disk/	odpwd(1)
default directory used by optical	disk commands /set the current	odcd(1)
default directory used by optical	disk commands /the current	odpwd(1)
cpflop(1) copy floppy	disk	cpflop(1)
disktab(4)	disk description file	disktab(4)
odls(1) list contents of optical	disk directories	odls(1)
odmkdir(1) create optical	disk directories	odmkdir(1)
odrmdir(1) delete optical	disk directories	odrmdir(1)
name with an existing optical	disk file /associate another	odln(1)
introduction to the optical	disk file system odintro(1)	odintro(1)
odcp(1) copy optical	disk files	odcp(1)
odrm(1) delete optical	disk files	odrm(1)
/change the file group of optical	disk files or directories	odchgrp(1)
the file protection of optical	disk files or directories /change	odchmod(1)
/change file ownership of optical	disk files or directories	odchown(1)
odmv(1) rename optical	disk files or directories	odmv(1)
fr_flop(1) continuous floppy	disk filters to_flop(1)	to_flop(1)
format(1) floppy	disk formatting utility	format(1)
ffsinode(4) structure of an FFS	disk i-node	ffsinode(4)

grabber	fg_dealloc(3A) deallocate a frame	fg_dealloc(3A)
entry /setgrent(3C) endgrent(3C)	fgetgrent(3C) get group file	getgrent(3C)
entry /setpwent(3C) endpwent(3C)	fgetpwent(3C) get password file	getpwent(3C)
frame buffer	fg_fbmode(3A) set the mode of a	fg_fbmode(3A)
frame buffer	fg_fbstat(3A) get the mode of a	fg_fbstat(3A)
the lookup tables of a frame/	fg_lut_in(3A) fg_lut_out(3A) load	fg_lut_in(3A)
tables of a frame/ fg_lut_in(3A)	fg_lut_out(3A) load the lookup	fg_lut_in(3A)
buffer output priority of the/	fg_priority(3A) determine frame	fg_priority(3A)
grabber to a known state	fg_reset(3A) force the frame	fg_reset(3A)
grabber window size	fg_size(3A) determine the frame	fg_size(3A)
select the video signal types/	fg_video_in(3A) fg_video_out(3A)	fg_video_in(3A)
signal types for/ fg_video_in(3A)	fg_video_out(3A) select the video	fg_video_in(3A)
start and stop video in a window	fg_viw_start(3A) fg_viw_stop(3A)	fg_viw_start(3A)
video in a/ fg_viw_start(3A)	fg_viw_stop(3A) start and stop	fg_viw_start(3A)
cpio(1) copy	file archives in and out	cpio(1)
rcs(1) change RCS	file attributes	rcs(1)
certification documentation	file /Intergraph software	certnote.com(4)
fcntl(2)	file control	fcntl(2)
fcntl(5)	file control options	fcntl(5)
rcp(1) remote	file copy	rcp(1)
core(4) format of core image	file	core(4)
FORTRAN/ fnum(3F) return the	file descriptor associated with a	fnum(3F)
logical unit associated with a	file descriptor /return FORTRAN	fdtounit(3F)
disktab(4) disk description	file	disktab(4)
fgetgrent(3C) get group	file entry /endgrent(3C)	getgrent(3C)
fgetpwent(3C) get password	file entry /endpwent(3C)	getpwent(3C)
error log configuration	file errord.rc(4)	errord.rc(4)
netex(1) DNP remote	file execution utility	netex(1)
software delivery documentation	file fixes.com(4) Intergraph	fixes.com(4)
aliases(4) aliases	file for sendmail(1M)	aliases(4)
intro(4) introduction to	file formats	intro(4)
status daemon directory and	file formats /recover(4) state(4)	statmon(4)
sccstorcs(1) build RCS	file from SCCS file	sccstorcs(1)
or/ odchgrp(1) change the	file group of optical disk files	odchgrp(1)
group(4) group	file	group(4)
limits(4)	file header for/	limits(4)
bootheader(4) boot	file header format	bootheader(4)
disk jukebox configuration	file JBCFG(4) optical	jbcfg(4)
kbmap(4) keyboard map	file	kbmap(4)
fmu(1) network	file management utility	fmu(1)
merge(1) three-way	file merge	merge(1)
the database for the mail aliases	file newaliases(1) rebuild	newaliases(1)
with an existing optical disk	file /associate another name	odln(1)
files or/ odchown(1) change	file ownership of optical disk	odchown(1)
passwd(4) password	file	passwd(4)
files or/ odchmod(1) change the	file protection of optical disk	odchmod(1)
rcsfile(4) format of RCS	file	rcsfile(4)
read(2) read from	file	read(2)
information for a common object	file reloc(4) relocation	reloc(4)

rename(2B) change the name of a	file	rename(2B)
build RCS file from SCCS	file sccstorcs(1)	sccstorcs(1)
XNS server information	file server.dat(4)	server.dat(4)
disk standalone configuration	file STANDCFG(4) optical	standcfg(4)
lstat(2B) get	file status	lstat(2B)
make a symbolic link to a	file symlink(2B)	symlink(2B)
backup(1) incremental	file system backup	backup(1)
mount(2) mount a	file system	mount(2)
npumount(1) mount and unmount	file system npmount(1)	npmount(1)
introduction to the optical disk	file system odintro(1)	odintro(1)
restore(1) incremental	file system restore	restore(1)
fstab(4)	file system table	fstab(4)
mnttab(4) mounted	file system table	mnttab(4)
rmtab(4) remotely mounted NFS	file system table	rmtab(4)
ffsfs(4) format of	file system volume	ffsfs(4)
exports(4) NFS	file systems being exported	exports(4)
ftruncate(2B) truncate a	file to a specified length	ftruncate(2B)
kermit(1) kermit	file transfer	kermit(1)
ftp(1) ARPANET	file transfer program	ftp(1)
tftp(1) trivial	file transfer program	tftp(1)
write(2) write to a	file	write(2)
and print process accounting	files acctcom(1) search	acctcom(1)
find(1) find	files	find(1)
fmu_receive(3N) receive	files from a remote system	fmu_receive(3N)
ident(1) identify	files	ident(1)
link editor for common object	files ld(1)	ld(1)
ln(1) link	files	ln(1)
that moves or renames one or more	files netmv(1) DNP command	netmv(1)
netrm(1) DNP command that removes	files	netrm(1)
odcp(1) copy optical disk	files	odcp(1)
odrm(1) delete optical disk	files	odrm(1)
odt(1) examine and modifies	files	odt(1)
netlpr(1) DNP command to print	file(s) on remote printers	netlpr(1)
the file group of optical disk	files or directories /change	odchgrp(1)
file protection of optical disk	files or directories /change the	odchmod(1)
file ownership of optical disk	files or directories /change	odchown(1)
odmv(1) rename optical disk	files or directories	odmv(1)
rm(1) rmdir(1) remove	files or directories	rm(1)
rcsclean(1) clean up working	files	rcsclean(1)
and other information about RCS	files rlog(1) print log messages	rlog(1)
CRM utility for monitoring open	files showfiles(1)	showfiles(1)
fmu_send(3N) send	files to a remote system	fmu_send(3N)
fr_flop(1) continuous floppy disk	filters to_flop(1)	to_flop(1)
find(1)	find files	find(1)
	find(1) find files	find(1)
delivery documentation file	fixes.com(4) Intergraph software	fixes.com(4)
/xio_clref(3A) xio_setef(3A) event	flag mask functions	xio_readef(3A)
allocate/deallocate an event	flag number /xio_deallocef(3A)	xio_allocef(3A)
cpflop(1) copy	floppy disk	cpflop(1)

getpwent(3C) getpwuid(3C)	getpwnam(3C) setpwent(3C)/	getpwent(3C)
setpwent(3C)/ getpwent(3C)	getpwuid(3C) getpwnam(3C)	getpwent(3C)
getrpcbynumber(3R)/ getrpcent(3R)	getrpcbyname(3R)	getrpcent(3R)
getrpcent(3R) getrpcbyname(3R)	getrpcbynumber(3R) setrpcent(3R)/	getrpcent(3R)
getrpcbynumber(3R) setrpcent(3R)/	getrpcent(3R) getrpcbyname(3R)	getrpcent(3R)
number	getrpcport(3R) get RPC port	getrpcport(3R)
getservent(3B) getservbyport(3B)	getservbyname(3B) setservent(3B)/	getservent(3B)
getservbyname(3B)/ getservent(3B)	getservbyport(3B)	getservent(3B)
getservbyname(3B) setservent(3B)/	getservent(3B) getservbyport(3B)	getservent(3B)
/setdomainname(3R)	get/set name of current domain	getdomainname(3R)
gethostname(2B) sethostname(2B)	get/set name of current host	gethostname(2B)
gethostid(2B) sethostid(2B)	get/set unique identifier of/	gethostid(2B)
getitimer(2B) setitimer(2B)	get/set value of interval timer	getitimer(2B)
	getsockname(2B) get socket name	getsockname(2B)
and set options on sockets	getsockopt(2B) setsockopt(2B) get	getsockopt(2B)
time	gettimeofday(2B) get date and	gettimeofday(2B)
all outstanding requests on a	GPIB channel /cancel	gpib_cancel(3A)
gpib_close(3A) close a	GPIB channel	gpib_close(3A)
send commands to a	GPIB channel /gpib_cmd_nw(3A)	gpib_cmd(3A)
issue a local lockout to a	GPIB channel gpib_lockout(3A)	gpib_lockout(3A)
gpib_open(3A) open a	GPIB channel	gpib_open(3A)
perform a parallel poll of a	GPIB channel gpib_ppreq(3A)	gpib_ppreq(3A)
conduct an IFC operation on a	GPIB channel gpib_reset(3A)	gpib_reset(3A)
gpib_remote(3A) put a	GPIB channel in a remote state	gpib_remote(3A)
gpib_clear(3A) clear a	GPIB channel or device	gpib_clear(3A)
the parallel poll response of a	GPIB device /configure	gpib_ppconf(3A)
gpib_read_nw(3A) read data from a	GPIB device gpib_read(3A)	gpib_read(3A)
conduct a serial poll of a	GPIB device gpib_spreq(3A)	gpib_spreq(3A)
gpib_trigger(3A) trigger a	GPIB device	gpib_trigger(3A)
gpib_write_nw(3A) write data to a	GPIB device gpib_write(3A)	gpib_write(3A)
gpib_local(3A) return a	GPIB device to local control	gpib_local(3A)
gpib_ppuconf(3A) unconfigure a	GPIB device's parallel poll/	gpib_ppuconf(3A)
/request notification for a	GPIB SRQ condition	gpib_service(3A)
outstanding requests on a GPIB/	gpib_cancel(3A) cancel all	gpib_cancel(3A)
channel or device	gpib_clear(3A) clear a GPIB	gpib_clear(3A)
channel	gpib_close(3A) close a GPIB	gpib_close(3A)
commands to a GPIB channel	gpib_cmd(3A) gpib_cmd_nw(3A) send	gpib_cmd(3A)
a GPIB channel gpib_cmd(3A)	gpib_cmd_nw(3A) send commands to	gpib_cmd(3A)
device to local control	gpib_local(3A) return a GPIB	gpib_local(3A)
lockout to a GPIB channel	gpib_lockout(3A) issue a local	gpib_lockout(3A)
	gpib_open(3A) open a GPIB channel	gpib_open(3A)
parallel poll response of a GPIB/	gpib_ppconf(3A) configure the	gpib_ppconf(3A)
poll of a GPIB channel	gpib_ppreq(3A) perform a parallel	gpib_ppreq(3A)
GPIB device's parallel poll/	gpib_ppuconf(3A) unconfigure a	gpib_ppuconf(3A)
read data from a GPIB device	gpib_read(3A) gpib_read_nw(3A)	gpib_read(3A)
GPIB device gpib_read(3A)	gpib_read_nw(3A) read data from a	gpib_read(3A)
channel in a remote state	gpib_remote(3A) put a GPIB	gpib_remote(3A)
operation on a GPIB channel	gpib_reset(3A) conduct an IFC	gpib_reset(3A)
gpib_service_nw(3A) request/	gpib_service(3A)	gpib_service(3A)

notification/ gpib_service(3A)	gpib_service_nw(3A) request	gpib_service(3A)
poll of a GPIB device	gpib_spreq(3A) conduct a serial	gpib_spreq(3A)
device	gpib_trigger(3A) trigger a GPIB	gpib_trigger(3A)
write data to a GPIB device	gpib_write(3A) gpib_write_nw(3A)	gpib_write(3A)
GPIB device gpib_write(3A)	gpib_write_nw(3A) write data to a	gpib_write(3A)
fg_alloc(3A) allocate a frame	grabber	fg_alloc(3A)
the output signal of the frame	grabber fg_blank(3A) blank	fg_blank(3A)
fg_dealloc(3A) deallocate a frame	grabber	fg_dealloc(3A)
load the lookup tables of a frame	grabber /fg_lut_out(3A)	fg_lut_in(3A)
output priority of the frame	grabber /determine frame buffer	fg_priority(3A)
fg_reset(3A) force the frame	grabber to a known state	fg_reset(3A)
fg_size(3A) determine the frame	grabber window size	fg_size(3A)
chown(1) chgrp(1) change owner or	group	chown(1)
endgrent(3C) fgetgrent(3C) get	group file entry /setgrent(3C)	getgrent(3C)
group(4)	group file	group(4)
getpgrp2(2B) get process	group	getpgrp2(2B)
setpgrp(2) set process	group ID	setpgrp(2)
send signal to a process	group killpg(2B)	killpg(2B)
odchgrp(1) change the file	group of optical disk files or/	odchgrp(1)
setpgrp2(2B) set process	group	setpgrp2(2B)
	group(4) group file	group(4)
/translation table to	handle long map names	ypmapxlate(4)
reset a process's priority after	handling a signal ucprelse(2I)	ucprelse(2I)
qpr(1) submit a	hardcopy print request to NQS	qpr(1)
sysid(2I) get the system	hardware identification number	sysid(2I)
csi_reset(3A) reset	hardware on CSI port	csi_reset(3A)
limits(4) file	header for/	limits(4)
bootheader(4) boot file	header format	bootheader(4)
diskpar(4) disk partition	header format	diskpar(4)
floppypar(4) partitioned floppy	header format	floppypar(4)
/ntohs(3B) convert values between	host and network byte order	byteorder(3B)
endhostent(3B) get network	host entry /sethostent(3B)	gethostbyname(3B)
hosts.equiv(4)	host equivalency name database	hosts.equiv(4)
unique identifier of current	host /sethostid(2B) get/set	gethostid(2B)
get/set name of current	host /sethostname(2B)	gethostname(2B)
hosts(4)	host name database	hosts(4)
and shell strategy for the local	host /show supported batch limits	qlimit(1)
the local/ ruptime(1) show	host status for each machine on	ruptime(1)
set or print name of current	host system hostname(1)	hostname(1)
current host system	hostname(1) set or print name of	hostname(1)
	hosts(4) host name database	hosts(4)
name database	hosts.equiv(4) host equivalency	hosts.equiv(4)
ntohs(3B) convert/ byteorder(3B)	htonl(3B) htons(3B) ntohl(3B)	byteorder(3B)
convert/ byteorder(3B) htonl(3B)	htons(3B) ntohl(3B) ntohs(3B)	byteorder(3B)
setpgrp(2) set process group	ID	setpgrp(2)
	ident(1) identify files	ident(1)
sysid(2I) get the system hardware	identification number	sysid(2I)
return CLIPPER processor	identifier getcpuid(2I)	getcpuid(2I)
/sethostid(2B) get/set unique	identifier of current host	gethostid(2B)

ident(1)	identify files	ident(1)
gpib_reset(3A) conduct an	IFC operation on a GPIB channel	gpib_reset(3A)
core(4) format of core	image file	core(4)
load an FPE coprocessor	image fpe_did_load(3A)	fpe_did_load(3A)
unload an FPE coprocessor	image fpe_did_unload(3A)	fpe_did_unload(3A)
limits(4) file header for	implementation-specific constants	limits(4)
backup(4) dumpdates(4)	incremental dump format	backup(4)
backup(1)	incremental file system backup	backup(1)
restore(1)	incremental file system restore	restore(1)
operations string(3B)	index(3B) rindex(3B) string	string(3B)
inet_network(3B) inet_ntoa(3B)/	inet(3B) inet_addr(3B)	inet(3B)
inet_ntoa(3B)/ inet(3B)	inet_addr(3B) inet_network(3B)	inet(3B)
/inet_ntoa(3B) inet_makeaddr(3B)	inet_lnaof(3B) inet_netof(3B)/	inet(3B)
/inet_network(3B) inet_ntoa(3B)	inet_makeaddr(3B) inet_lnaof(3B)/	inet(3B)
/inet_makeaddr(3B) inet_lnaof(3B)	inet_netof(3B) Internet address/	inet(3B)
inet(3B) inet_addr(3B)	inet_network(3B) inet_ntoa(3B)/	inet(3B)
/inet_addr(3B) inet_network(3B)	inet_ntoa(3B) inet_makeaddr(3B)/	inet(3B)
/print log messages and other	information about RCS files	rlog(1)
server.dat(4) XNS server	information file	server.dat(4)
file reloc(4) relocation	information for a common object	reloc(4)
getavailsmem(2I) return memory	information /getfreemem(2I)	getmemsize(2I)
readinfo(2I) read system activity	information	readinfo(2I)
f77initio(3F) f77uninitio(3F)	initialize or terminate FORTRAN/	f77initio(3F)
connect(2B)	initiate a connection on a socket	connect(2B)
structure of an FFS disk	i-node ffsinode(4)	ffsinode(4)
/report number of free blocks and	i-nodes on an optical volume	oddf(1)
readv(2B) read	input from a socket	readv(2B)
queue insque(3B) remque(3B)	insert/remove element from a	insque(3B)
insert/remove element from a/	insque(3B) remque(3B)	insque(3B)
ypprot_err(3R) YP client	interfac /yperr_string(3R)	ypclnt(3R)
telnet(1) user	interface to the TELNET protocol	telnet(1)
/inet_lnaof(3B) inet_netof(3B)	Internet address manipulation/	inet(3B)
csh(1) a shell (command	interpreter) with C-like syntax	csh(1)
setitimer(2B) get/set value of	interval timer getitimer(2B)	getitimer(2B)
and application programs	intro(1) introduction to commands	intro(1)
calls and error numbers	intro(2) introduction to system	intro(2)
functions and libraries	intro(3) introduction to	intro(3)
synchronous/asynchronous I/O/	intro(3A) introduction to the	intro(3A)
library functions	intro(3B) introduction to BSD	intro(3B)
Intergraph communications/	intro(3N) introduction to	intro(3N)
RPC/XDR/YP service functions and/	intro(3R) introduction to	intro(3R)
formats	intro(4) introduction to file	intro(4)
miscellaneous facilities	intro(5) introduction to	intro(5)
functions intro(3B)	introduction to BSD library	intro(3B)
application programs intro(1)	introduction to commands and	intro(1)
intro(4)	introduction to file formats	intro(4)
libraries intro(3)	introduction to functions and	intro(3)
communications/ intro(3N)	introduction to Intergraph	intro(3N)
facilities intro(5)	introduction to miscellaneous	intro(5)

rwho(1) lists users logged in to	machines on the local network	rwho(1)
program mt(1)	magnetic tape manipulation	mt(1)
ansitape(1) ANSI-standard	magtape label program	ansitape(1)
ansitape(4) ANSI standard	magtape labels	ansitape(4)
rebuild the database for the	mail aliases file newaliases(1)	newaliases(1)
mailq(1) display a listing of the	mail queue used by sendmail(1M)	mailq(1)
mailstats(1) display	mail statistics	mailstats(1)
cumail(1) DNP	mail transport program	cumail(1)
mail queue used by sendmail(1M)	mailq(1) display a listing of the	mailq(1)
statistics	mailstats(1) display mail	mailstats(1)
adep(1) Ada program	makefile generator	adep(1)
manual	man(1) print entries in this	man(1)
Intergraph network clearinghouse	management program clh(1)	clh(1)
sigignore(2) sigpause(2) signal	management /sigrelse(2)	sigset(2)
fmu(1) network file	management utility	fmu(1)
mt(1) magnetic tape	manipulation program	mt(1)
rtape(1) remote tape	manipulation program	rtape(1)
/inet_netof(3B) Internet address	manipulation routines	inet(3B)
man(1) print entries in this	manual	man(1)
kbmap(4) keyboard	map file	kbmap(4)
translation table to handle long	map names ypmapxlate(4)	ypmapxlate(4)
of one or more keys from a YP	map ypmatch(1) print the value	ypmatch(1)
xio_setef(3A) event flag	mask functions /xio_clref(3A)	xio_readef(3A)
master(4)	master configuration database	master(4)
database	master(4) master configuration	master(4)
spawn a new process in a virtual	memory efficient way vfork(2B)	vfork(2B)
/getavailsmem(2I) return	memory information	getmemsize(2I)
read data from a PDI port into	memory /pdi_read_nw(3A)	pdi_read(3A)
CRM utility for monitoring	memory regions monregion(1)	monregion(1)
utility for monitoring process	memory regions showmemory(1) CRM	showmemory(1)
RLE from pipe to	memory /rle_pipe_mem_nw(3A)	rle_pipe_mem(3A)
transfer data from scanner to	memory /sif_scan_mem_nw(3A)	sif_scan_mem(3A)
/pdi_write_nw(3A) write data from	memory to a PDI port	pdi_write(3A)
/transfer data from	memory to pipe	sif_mem_pipe(3A)
monitoring physical and virtual	memory topmem(1) CRM utility for	topmem(1)
vlock(2I) lock an area of	memory	vlock(2I)
vunlock(2I) unlock an area of	memory	vunlock(2I)
merge(1) three-way file	merge	merge(1)
rcsmerge(1)	merge RCS revisions	rcsmerge(1)
	merge(1) three-way file merge	merge(1)
/recvmsg(2B) receive a	message from a socket	recv(2B)
sendto(2B) sendmsg(2B) send a	message from a socket send(2B)	send(2B)
local XNS/ netmsg(1) send a	message to console devices on the	netmsg(1)
about RCS/ rlog(1) print log	messages and other information	rlog(1)
intro(5) introduction to	miscellaneous facilities	intro(5)
library	mkshlib(1) create a shared	mkshlib(1)
table	mnttab(4) mounted file system	mnttab(4)
chmod(1) change	mode	chmod(1)
fg_fbmode(3A) set the	mode of a frame buffer	fg_fbmode(3A)

fg_fbstat(3A) get the	mode of a frame buffer	fg_fbstat(3A)
/aux_modem_nw(3A) get	modem change from a serial port	aux_modem(3A)
port aux_cancel_modem(3A) cancel	modem change state on a serial	aux_cancel_modem(3A)
fmu_set(3N) set FMU	modes	fmu_set(3N)
odt(1) examine and	modifies files	odt(1)
option sigcld(2I)	modify SIGCLD on stop signal	sigcld(2I)
crm(1) CLIX Resource	Monitor	crm(1)
monproc(1) CRM utility for	monitoring a process	monproc(1)
topcpu(1) CRM utility for	monitoring CPU time	topcpu(1)
topio(1) CRM utility for	monitoring I/O activity	topio(1)
monregion(1) CRM utility for	monitoring memory regions	monregion(1)
showfiles(1) CRM utility for	monitoring open files	showfiles(1)
topfault(1) CRM utility for	monitoring page faults	topfault(1)
memory topmem(1) CRM utility for	monitoring physical and virtual	topmem(1)
showmemory(1) CRM utility for	monitoring process memory regions	showmemory(1)
topsys(1) CRM utility for	monitoring system activity	topsys(1)
watcher(1) CRM utility for	monitoring system calls and/	watcher(1)
monparam(1) CRM utility for	monitoring system parameters	monparam(1)
monitoring system parameters	monparam(1) CRM utility for	monparam(1)
monitoring a process	monproc(1) CRM utility for	monproc(1)
monitoring memory regions	monregion(1) CRM utility for	monregion(1)
mount(2)	mount a file system	mount(2)
npmount(1) npumount(1)	mount and unmount file system	npmount(1)
	mount(2) mount a file system	mount(2)
mnttab(4)	mounted file system table	mnttab(4)
rmtab(4) remotely	mounted NFS file system table	rmtab(4)
pdi_ifb(3A) pdi_ifb_nw(3A)	move data from a PDI port to a/	pdi_ifb(3A)
files netmv(1) DNP command that	moves or renames one or more	netmv(1)
dtu(1) utd(1) copy between	MS-DOS and CLIX	dtu(1)
dls(1) list contents of	MS-DOS directory	dls(1)
program	mt(1) magnetic tape manipulation	mt(1)
I/O cpio(1) scpio(1)	multibuffering and asynchronous	scpio(1)
select(2B) synchronous I/O	multiplexing	select(2B)
hosts(4) host	name database	hosts(4)
hosts.equiv(4) host equivalency	name database	hosts.equiv(4)
networks(4) network	name database	networks(4)
protocols(4) protocol	name database	protocols(4)
services(4) service	name database	services(4)
getsockname(2B) get socket	name	getsockname(2B)
rename(2B) change the	name of a file	rename(2B)
getpeername(2B) get	name of connected peer	getpeername(2B)
/setdomainname(3R) get/set	name of current domain	getdomainname(3R)
/sethostname(2B) get/set	name of current host	gethostname(2B)
hostname(1) set or print	name of current host system	hostname(1)
domname(1) set or display	name of current YP domain	domname(1)
setnodename(2I) set new node	name	setnodename(2I)
bind(2B) bind a	name to a socket	bind(2B)
disk/ odln(1) associate another	name with an existing optical	odln(1)
table to handle long map	names ypmapxlate(4) translation	ypmapxlate(4)

allocate/deallocate an event flag	number /xio_deallocef(3A)	xio_allocef(3A)
to system calls and error	numbers intro(2) introduction	intro(2)
clh_vbyop(3N) lookup value by	object and property	clh_vbyop(3N)
information for a common	object file reloc(4) relocation	reloc(4)
ld(1) link editor for common	object files	ld(1)
directory used by optical disk/	odcd(1) set the current default	odcd(1)
of optical disk files or/	odchgrp(1) change the file group	odchgrp(1)
protection of optical disk files/	odchmod(1) change the file	odchmod(1)
of optical disk files or/	odchown(1) change file ownership	odchown(1)
	odcp(1) copy optical disk files	odcp(1)
blocks and i-nodes on an optical/	oddf(1) report number of free	oddf(1)
optical disk file system	odintro(1) introduction to the	odintro(1)
with an existing optical disk/	odln(1) associate another name	odln(1)
disk directories	odls(1) list contents of optical	odls(1)
directories	odmkdir(1) create optical disk	odmkdir(1)
or directories	odmv(1) rename optical disk files	odmv(1)
default directory used by/	odpwd(1) display the current	odpwd(1)
	odrm(1) delete optical disk files	odrm(1)
directories	odrmdir(1) delete optical disk	odrmdir(1)
	odt(1) examine and modifies files	odt(1)
lpr(1) BSD	offline print	lpr(1)
cnv_open(3A)	open a CNV channel	cnv_open(3A)
csi_open(3A)	open a CSI port	csi_open(3A)
gpib_open(3A)	open a GPIB channel	gpib_open(3A)
pdi_open(3A)	open a PDI port	pdi_open(3A)
aux_open(3A)	open a serial port	aux_open(3A)
sif_open(3A)	open a SIF channel	sif_open(3A)
nlf_open(3A)	open an NLF channel	nlf_open(3A)
rle_open(3A)	open an RLE channel	rle_open(3A)
CRM utility for monitoring	open files showfiles(1)	showfiles(1)
gpib_reset(3A) conduct an IFC	operation on a GPIB channel	gpib_reset(3A)
ffs(3B) bit and byte string	operations /bcmp(3B) bzero(3B)	bstring(3B)
index(3B) rindex(3B) string	operations string(3B)	string(3B)
current default directory used by	optical disk commands /set the	odcd(1)
current default directory used by	optical disk commands /the	odpwd(1)
odls(1) list contents of	optical disk directories	odls(1)
odmkdir(1) create	optical disk directories	odmkdir(1)
odrmdir(1) delete	optical disk directories	odrmdir(1)
another name with an existing	optical disk file /associate	odln(1)
odintro(1) introduction to the	optical disk file system	odintro(1)
odcp(1) copy	optical disk files	odcp(1)
odrm(1) delete	optical disk files	odrm(1)
/change the file group of	optical disk files or directories	odchgrp(1)
/change the file protection of	optical disk files or directories	odchmod(1)
/change file ownership of	optical disk files or directories	odchown(1)
odmv(1) rename	optical disk files or directories	odmv(1)
configuration file JBCFG(4)	optical disk jukebox	jbcfg(4)
configuration file STANDCFG(4)	optical disk standalone	standcfg(4)
of free blocks and i-nodes on an	optical volume /report number	oddf(1)

modify SIGCLD on stop signal	option sigcld(2I)	sigcld(2I)
fcntl(5) file control	options	fcntl(5)
stty(1) set the	options for a terminal	stty(1)
/setsockopt(2B) get and set	options on sockets	getsockopt(2B)
between host and network byte	order /ntohs(3B) convert values	byteorder(3B)
common assembler and link editor	output a.out(4)	a.out(4)
logical unit flush(3F) flush the	output for the specified FORTRAN	flush(3F)
grabber /determine frame buffer	output priority of the frame	fg_priority(3A)
grabber fg_blank(3A) blank the	output signal of the frame	fg_blank(3A)
writev(2B) write	output to a socket	writev(2B)
CSI port csi_cancel(3A) cancel	outstanding asynchronous I/O on a	csi_cancel(3A)
PDI port pdi_cancel(3A) cancel	outstanding asynchronous I/O on a	pdi_cancel(3A)
SIF/ sif_cancel(3A) cancel	outstanding asynchronous I/O on a	sif_cancel(3A)
an RLE/ rle_cancel(3A) cancel	outstanding asynchronous I/O on	rle_cancel(3A)
aux_cancel(3A) cancel	outstanding read on a serial port	aux_cancel(3A)
gpib_cancel(3A) cancel all	outstanding requests on a GPIB/	gpib_cancel(3A)
chown(1) chgrp(1) change	owner or group	chown(1)
or/ odchown(1) change file	ownership of optical disk files	odchown(1)
/csi_cmd_nw(3A) send command	packets to a CSI port	csi_cmd(3A)
CRM utility for monitoring	page faults topfault(1)	topfault(1)
getpagesize(2B) get system	page size	getpagesize(2B)
socketpair(2B) create a	pair of connected sockets	socketpair(2B)
gpib_ppreq(3A) perform a	parallel poll of a GPIB channel	gpib_ppreq(3A)
/unconfigure a GPIB device's	parallel poll response	gpib_ppuconf(3A)
gpib_ppconf(3A) configure the	parallel poll response of a GPIB/	gpib_ppconf(3A)
send a control word to the	parallel port /plt_ctrl_nw(3A)	plot_ctrl(3A)
send a control word to the ROP	parallel port /rplt_ctrl_nw(3A)	plot_ctrl_rop(3A)
plt_data_nw(3A) write data to the	parallel port /plt_data(3A)	plot_data(3A)
write data to the ROP	parallel port /rplt_data_nw(3A)	plot_data_rop(3A)
pdi_setup(3A) establish	parameters for a PDI port	pdi_setup(3A)
sif_setup(3A) establish	parameters for a SIF channel	sif_setup(3A)
rle_setup(3A) establish	parameters for an RLE channel	rle_setup(3A)
CRM utility for monitoring system	parameters monparam(1)	monparam(1)
shutdown(2B) shut down	part of a full-duplex connection	shutdown(2B)
diskpar(4) disk	partition header format	diskpar(4)
floppypar(4)	partitioned floppy header format	floppypar(4)
pc(1)	Pascal compiler	pc(1)
	passwd(4) password file	passwd(4)
endpwent(3C) fgetpwent(3C) get	password file entry /setpwent(3C)	getpwent(3C)
passwd(4)	password file	passwd(4)
yppasswd(1) change login	password in YP	yppasswd(1)
yppasswd(3R) update user	password in YP	yppasswd(3R)
generate line printer ripple	pattern lptest(1)	lptest(1)
	pc(1) Pascal compiler	pc(1)
/pdi_read_nw(3A) read data from a	PDI port into memory	pdi_read(3A)
outstanding asynchronous I/O on a	PDI port pdi_cancel(3A) cancel	pdi_cancel(3A)
pdi_close(3A) close a	PDI port	pdi_close(3A)
pdi_open(3A) open a	PDI port	pdi_open(3A)
establish parameters for a	PDI port pdi_setup(3A)	pdi_setup(3A)

write data from memory to a	PDI port /pdi_write_nw(3A)	pdi_write(3A)
pdi_ifb_nw(3A) move data from a	PDI port to a window pdi_ifb(3A)	pdi_ifb(3A)
asynchronous I/O on a PDI port	pdi_cancel(3A) cancel outstanding	pdi_cancel(3A)
	pdi_close(3A) close a PDI port	pdi_close(3A)
data from a PDI port to a window	pdi_ifb(3A) pdi_ifb_nw(3A) move	pdi_ifb(3A)
PDI port to a window pdi_ifb(3A)	pdi_ifb_nw(3A) move data from a	pdi_ifb(3A)
	pdi_open(3A) open a PDI port	pdi_open(3A)
data from a PDI port into memory	pdi_read(3A) pdi_read_nw(3A) read	pdi_read(3A)
PDI port into/ pdi_read(3A)	pdi_read_nw(3A) read data from a	pdi_read(3A)
parameters for a PDI port	pdi_setup(3A) establish	pdi_setup(3A)
write data from memory to a PDI/	pdi_write(3A) pdi_write_nw(3A)	pdi_write(3A)
memory to a PDI/ pdi_write(3A)	pdi_write_nw(3A) write data from	pdi_write(3A)
get name of connected	peer getpeername(2B)	getpeername(2B)
alignment trap disposition	per executable /set/report	aligntrap(1)
channel gpib_ppreq(3A)	perform a parallel poll of a GPIB	gpib_ppreq(3A)
/CRM utility for monitoring	physical and virtual memory	topmem(1)
rpipe(1) remote	pipe program	rpipe(1)
transfer data from memory to	pipe /sif_mem_pipe_nw(3A)	sif_mem_pipe(3A)
transfer data from scanner to	pipe /sif_scan_pipe_nw(3A)	sif_scan_pipe(3A)
rle_pipe_mem_nw(3A) RLE from	pipe to memory rle_pipe_mem(3A)	rle_pipe_mem(3A)
plt_ctrl_nw(3A) send a control/	plot_ctrl(3A) plt_ctrl(3A)	plot_ctrl(3A)
rplt_ctrl_nw(3A) send a control/	plot_ctrl_rop(3A) rplt_ctrl(3A)	plot_ctrl_rop(3A)
plt_data_nw(3A) write data to/	plot_data(3A) plt_data(3A)	plot_data(3A)
rplt_data_nw(3A) write data to/	plot_data_rop(3A) rplt_data(3A)	plot_data_rop(3A)
a control word to/ plot_ctrl(3A)	plt_ctrl(3A) plt_ctrl_nw(3A) send	plot_ctrl(3A)
word/ plot_ctrl(3A) plt_ctrl(3A)	plt_ctrl_nw(3A) send a control	plot_ctrl(3A)
write data to the/ plot_data(3A)	plt_data(3A) plt_data_nw(3A)	plot_data(3A)
plot_data(3A) plt_data(3A)	plt_data_nw(3A) write data to the/	plot_data(3A)
gpib_ppreq(3A) perform a parallel	poll of a GPIB channel	gpib_ppreq(3A)
gpib_spreq(3A) conduct a serial	poll of a GPIB device	gpib_spreq(3A)
a GPIB device's parallel	poll response /unconfigure	gpib_ppuconf(3A)
/configure the parallel	poll response of a GPIB device	gpib_ppconf(3A)
generate a break on a serial	port aux_break(3A)	aux_break(3A)
outstanding read on a serial	port aux_cancel(3A) cancel	aux_cancel(3A)
modem change state on a serial	port aux_cancel_modem(3A) cancel	aux_cancel_modem(3A)
aux_close(3A) close a serial	port	aux_close(3A)
get modem change from a serial	port /aux_modem_nw(3A)	aux_modem(3A)
aux_open(3A) open a serial	port	aux_open(3A)
with error byte from a serial	port /aux_rawrd_nw(3A) read data	aux_rawrd(3A)
read data from a serial	port /aux_read_nw(3A)	aux_read(3A)
write data to a serial	port /aux_write_nw(3A)	aux_write(3A)
asynchronous I/O on a CSI	port /cancel outstanding	csi_cancel(3A)
a specific command on a CSI	port csi_ccan(3A) cancel	csi_ccan(3A)
csi_close(3A) close a CSI	port	csi_close(3A)
send command packets to a CSI	port csi_cmd(3A) csi_cmd_nw(3A)	csi_cmd(3A)
receive delayed status from a CSI	port csi_dstat_nw(3A)	csi_dstat_nw(3A)
csi_open(3A) open a CSI	port	csi_open(3A)
reset hardware on CSI	port csi_reset(3A)	csi_reset(3A)
status requests on a CSI	port /cancel unsolicited	csi_ucan(3A)

unsolicited status from a CSI	port /csi_ustat_nw(3A) receive	csi_ustat(3A)
csi_status(3A) read the CSI	port DR11 status lines	csi_status(3A)
read data from a PDI	port into memory /pdi_read_nw(3A)	pdi_read(3A)
getrpcport(3R) get RPC	port number	getrpcport(3R)
asynchronous I/O on a PDI	port /cancel outstanding	pdi_cancel(3A)
pdi_close(3A) close a PDI	port	pdi_close(3A)
pdi_open(3A) open a PDI	port	pdi_open(3A)
establish parameters for a PDI	port pdi_setup(3A)	pdi_setup(3A)
write data from memory to a PDI	port /pdi_write_nw(3A)	pdi_write(3A)
a control word to the parallel	port /plt_ctrl_nw(3A) send	plot_ctrl(3A)
control word to the ROP parallel	port /rplt_ctrl_nw(3A) send a	plot_ctrl_rop(3A)
write data to the parallel	port /plt_data_nw(3A)	plot_data(3A)
write data to the ROP parallel	port /rplt_data_nw(3A)	plot_data_rop(3A)
move data from a PDI	port to a window /pdi_ifb_nw(3A)	pdi_ifb(3A)
types(5)	primitive system data types	types(5)
man(1)	print entries in this manual	man(1)
netlpr(1) DNP command to	print file(s) on remote printers	netlpr(1)
information about RCS/ rlog(1)	print log messages and other	rlog(1)
lpr(1) BSD offline	print	lpr(1)
hostname(1) set or	print name of current host system	hostname(1)
acctcom(1) search and	print process accounting files	acctcom(1)
qpr(1) submit a hardcopy	print request to NQS	qpr(1)
keys from a YP map ypmatch(1)	print the value of one or more	ypmatch(1)
ypcat(1)	print values in a YP database	ypcat(1)
capability database	printcap(4) BSD printer	printcap(4)
printcap(4) BSD	printer capability database	printcap(4)
lptest(1) generate line	printer ripple pattern	lptest(1)
remove jobs from the BSD line	printer spooling queue lprm(1)	lprm(1)
to print file(s) on remote	printers netlpr(1) DNP command	netlpr(1)
ucprelse(2I) reset a process's	priority after handling a signal	ucprelse(2I)
ucppri(2I) check if a UCP	priority is already in use	ucppri(2I)
/determine frame buffer output	priority of the frame grabber	fg_priority(3A)
ucpsig(2I) set process to a UCP	priority on receipt of a signal	ucpsig(2I)
ucpclr(2I) clear process UCP	priority	ucpclr(2I)
ucpinq(2I) return the UCP	priority	ucpinq(2I)
ucpnice(1) run a process at UCP	priority	ucpnice(1)
ucpset(2I) set a process to a UCP	priority	ucpset(2I)
acctcom(1) search and print	process accounting files	acctcom(1)
ucpnice(1) run a	process at UCP priority	ucpnice(1)
setup for code execution in the	process data section exedata(2I)	exedata(2I)
exit(2) _exit(2) terminate	process	exit(2)
getpgrp2(2B) get	process group	getpgrp2(2B)
setpgrp(2) set	process group ID	setpgrp(2)
killpg(2B) send signal to a	process group	killpg(2B)
setpgrp2(2B) set	process group	setpgrp2(2B)
efficient/ vfork(2B) spawn a new	process in a virtual memory	vfork(2B)
/CRM utility for monitoring	process memory regions	showmemory(1)
CRM utility for monitoring a	process monproc(1)	monproc(1)
request/ xio_notify(3A) notify a	process of an asynchronous	xio_notify(3A)

receipt of a/ ucpsig(2I) set	process to a UCP priority on	ucpsig(2I)
ucpset(2I) set a	process to a UCP priority	ucpset(2I)
wait2(2I) wait for	process to terminate	wait2(2I)
wait3(2B) wait for	process to terminate	wait3(2B)
ucpclr(2I) clear	process UCP priority	ucpclr(2I)
getcpuid(2I) return CLIPPER	processor identifier	getcpuid(2I)
/clipper(1) ns32000(1) vax(1) get	processor type truth value	machid(1)
a signal ucprelse(2I) reset a	process's priority after handling	ucprelse(2I)
ANSI-standard magtape label	program ansitape(1)	ansitape(1)
ab(1) Ada	program beautifier	ab(1)
network clearinghouse management	program clh(1) Intergraph	clh(1)
cumail(1) DNP mail transport	program	cumail(1)
ftp(1) ARPANET file transfer	program	ftp(1)
BSD spool queue examination	program lpq(1)	lpq(1)
adep(1) Ada	program makefile generator	adep(1)
mt(1) magnetic tape manipulation	program	mt(1)
ncp(1) DNP network control	program	ncp(1)
rpc(4) RPC	program number database	rpc(4)
rpipe(1) remote pipe	program	rpipe(1)
rtape(1) remote tape manipulation	program	rtape(1)
connect to a server	program sni_connect(3N)	sni_connect(3N)
tftp(1) trivial file transfer	program	tftp(1)
visit(1) Intergraph remote login	program	visit(1)
the standard/restricted command	programming language /shell,	ksh(1)
to commands and application	programs intro(1) introduction	intro(1)
lookup value by object and	property clh_vbyop(3N)	clh_vbyop(3N)
or/ odchmod(1) change the file	protection of optical disk files	odchmod(1)
rpcgen(1) an RPC	protocol compiler	rpcgen(1)
endprotoent(3B) get	protocol entry /setprotoent(3B)	getprotoent(3B)
protocols(4)	protocol name database	protocols(4)
user interface to the TELNET	protocol telnet(1)	telnet(1)
RPC/XDR/YP service functions and	protocols /introduction to	intro(3R)
database	protocols(4) protocol name	protocols(4)
state gpib_remote(3A)	put a GPIB channel in a remote	gpib_remote(3A)
requests	qdel(1) delete or signal NQS	qdel(1)
devices	qdev(1) display the status of NQS	qdev(1)
limits and shell strategy for/	qlimit(1) show supported batch	qlimit(1)
request to NQS	qpr(1) submit a hardcopy print	qpr(1)
NQS queues	qstat(1) display the status of	qstat(1)
request	qsub(1) submit an NQS batch	qsub(1)
lpq(1) BSD spool	queue examination program	lpq(1)
insert/remove element from a	queue insque(3B) remque(3B)	insque(3B)
the BSD line printer spooling	queue lprm(1) remove jobs from	lprm(1)
/display a listing of the mail	queue used by sendmail(1M)	mailq(1)
display the status of NQS	queues qstat(1)	qstat(1)
random(3B) srandom(3B) better	random number generator	random(3B)
random number generator	random(3B) srandom(3B) better	random(3B)
dialect	ratfor(1) rational FORTRAN	ratfor(1)
ratfor(1)	rational FORTRAN dialect	ratfor(1)

for returning a stream to a	remote command /routines	rcmd(3B)
rexec(3B) return stream to a	remote command	rexec(3B)
rcp(1)	remote file copy	rcp(1)
netex(1) DNP	remote file execution utility	netex(1)
/connect/disconnect to	remote FMU server	fmu_connect(3N)
sethost(1) DNP	remote login DECnet or CLIX node.	sethost(1)
visit(1) Intergraph	remote login program	visit(1)
rlogin(1)	remote login	rlogin(1)
rpipe(1)	remote pipe program	rpipe(1)
DNP command to print file(s) on	remote printers netlpr(1)	netlpr(1)
put a GPIB channel in a	remote state gpib_remote(3A)	gpib_remote(3A)
execute the specified command on	remote system fmu_rcmd(3N)	fmu_rcmd(3N)
receive files from a	remote system fmu_receive(3N)	fmu_receive(3N)
fmu_send(3N) send files to a	remote system	fmu_send(3N)
lists the directory contents on a	remote system /DNP command that	netls(1)
rtc(1)	remote tape control	rtc(1)
/rtc_deallocate(3N) rtc_notify(3N)	remote tape control	rtc_allocate(3N)
rtape(1)	remote tape manipulation program	rtape(1)
.rhosts(4)	remote user access list	.rhosts(4)
table rmtab(4)	remotely mounted NFS file system	rmtab(4)
rm(1) rmdir(1)	remove files or directories	rm(1)
printer spooling queue lprm(1)	remove jobs from the BSD line	lprm(1)
netrm(1) DNP command that	removes files	netrm(1)
from a queue insque(3B)	remque(3B) insert/remove element	insque(3B)
directories odmv(1)	rename optical disk files or	odmv(1)
file	rename(2B) change the name of a	rename(2B)
/DNP command that moves or	renames one or more files	netmv(1)
errors(1) error logging	report generator	errors(1)
i-nodes on an optical/ oddf(1)	report number of free blocks and	oddf(1)
jukeboxes jbconfig(1)	report the configuration of the	jbconfig(1)
a process of an asynchronous	request completion /notify	xio_notify(3A)
SRQ/ /gpib_service_nw(3A)	request notification for a GPIB	gpib_service(3A)
qsub(1) submit an NQS batch	request	qsub(1)
fpe_cancel_dma(3A) cancel write	request to an FPE coprocessor	fpe_cancel_dma(3A)
qpr(1) submit a hardcopy print	request to NQS	qpr(1)
/cancel unsolicited status	requests on a CSI port	csi_ucan(3A)
/cancel all outstanding	requests on a GPIB channel	gpib_cancel(3A)
qdel(1) delete or signal NQS	requests	qdel(1)
handling a signal ucprelse(2I)	reset a process's priority after	ucprelse(2I)
csi_reset(3A)	reset hardware on CSI port	csi_reset(3A)
crm(1) CLIX	Resource Monitor	crm(1)
a GPIB device's parallel poll	response /unconfigure	gpib_ppuconf(3A)
/configure the parallel poll	response of a GPIB device	gpib_ppconf(3A)
incremental file system	restore restore(1)	restore(1)
system restore	restore(1) incremental file	restore(1)
control gpib_local(3A)	return a GPIB device to local	gpib_local(3A)
identifier getcpuid(2I)	return CLIPPER processor	getcpuid(2I)
associated with a/ fdtounit(3F)	return FORTRAN logical unit	fdtounit(3F)
/getfreemem(2I) getavailsmem(2I)	return memory information	getmemsize(2I)

setnodename(2I)	set new node name	setnodename(2I)
/setsockopt(2B) get and	set options on sockets	getsockopt(2B)
domain domname(1)	set or display name of current YP	domname(1)
system hostname(1)	set or print name of current host	hostname(1)
setpgrp(2)	set process group ID	setpgrp(2)
setpgrp2(2B)	set process group	setpgrp2(2B)
receipt of a signal ucpsig(2I)	set process to a UCP priority on	ucpsig(2I)
used by optical disk/ odcd(1)	set the current default directory	odcd(1)
fg_fbmode(3A)	set the mode of a frame buffer	fg_fbmode(3A)
stty(1)	set the options for a terminal	stty(1)
current domain getdomainname(3R)	setdomainname(3R) get/set name of	getdomainname(3R)
/getgrgid(3C) getgrnam(3C)	setgrent(3C) endgrent(3C)/	getgrent(3C)
DECnet or CLIX node.	sethost(1) DNP remote login	sethost(1)
/gethostbyaddr(3B) gethostent(3B)	sethostent(3B) endhostent(3B) get/	gethostbyname(3B)
identifier of/ gethostid(2B)	sethostid(2B) get/set unique	gethostid(2B)
current host gethostname(2B)	sethostname(2B) get/set name of	gethostname(2B)
interval timer getitimer(2B)	setitimer(2B) get/set value of	getitimer(2B)
/getnetbyaddr(3B) getnetbyname(3B)	setnetent(3B) endnetent(3B) get/	getnetent(3B)
	setnodename(2I) set new node name	setnodename(2I)
	setpgrp(2) set process group ID	setpgrp(2)
	setpgrp2(2B) set process group	setpgrp2(2B)
get protocol/ /getprotobyname(3B)	setprotoent(3B) endprotoent(3B)	getprotoent(3B)
/getpwuid(3C) getpwnam(3C)	setpwent(3C) endpwent(3C)/	getpwent(3C)
disposition per/ aligntrap(1)	set/report alignment trap	aligntrap(1)
RPC entry /getrpcbynumber(3R)	setrpcent(3R) endrpcent(3R) get	getrpcent(3R)
service entry /getservbyname(3B)	setservent(3B) endservent(3B) get	getservent(3B)
options on/ getsockopt(2B)	setsockopt(2B) get and set	getsockopt(2B)
process data section exedata(2I)	setup for code execution in the	exedata(2I)
mkshlib(1) create a	shared library	mkshlib(1)
C-like syntax csh(1) a	shell (command interpreter) with	csh(1)
/show supported batch limits and	shell strategy for the local host	qlimit(1)
command/ ksh(1) krsh(1)	shell, the standard/restricted	ksh(1)
on the local network ruptime(1)	show host status for each machine	ruptime(1)
shell strategy for the/ qlimit(1)	show supported batch limits and	qlimit(1)
monitoring open files	showfiles(1) CRM utility for	showfiles(1)
monitoring process memory/	showmemory(1) CRM utility for	showmemory(1)
connection shutdown(2B)	shut down part of a full-duplex	shutdown(2B)
full-duplex connection	shutdown(2B) shut down part of a	shutdown(2B)
outstanding asynchronous I/O on a	SIF channel /cancel	sif_cancel(3A)
sif_close(3A) close a	SIF channel	sif_close(3A)
sif_open(3A) open a	SIF channel	sif_open(3A)
establish parameters for a	SIF channel sif_setup(3A)	sif_setup(3A)
asynchronous I/O on a SIF/	sif_cancel(3A) cancel outstanding	sif_cancel(3A)
	sif_close(3A) close a SIF channel	sif_close(3A)
sif_mem_pipe_nw(3A) transfer/	sif_mem_pipe(3A)	sif_mem_pipe(3A)
from memory to/ sif_mem_pipe(3A)	sif_mem_pipe_nw(3A) transfer data	sif_mem_pipe(3A)
	sif_open(3A) open a SIF channel	sif_open(3A)
sif_scan_mem_nw(3A) transfer/	sif_scan_mem(3A)	sif_scan_mem(3A)
from scanner to/ sif_scan_mem(3A)	sif_scan_mem_nw(3A) transfer data	sif_scan_mem(3A)

sif_scan_pipe_nw(3A) transfer/	sif_scan_pipe(3A)	sif_scan_pipe(3A)
data from/ sif_scan_pipe(3A)	sif_scan_pipe_nw(3A) transfer	sif_scan_pipe(3A)
parameters for a SIF channel	sif_setup(3A) establish	sif_setup(3A)
sigcld(2I) modify	SIGCLD on stop signal option	sigcld(2I)
signal option	sigcld(2I) modify SIGCLD on stop	sigcld(2I)
sigignore(2)/ sigset(2)	sighold(2) sigrelse(2)	sigset(2)
sigset(2) sighold(2) sigrelse(2)	sigignore(2) sigpause(2) signal/	sigset(2)
sigignore(2) sigpause(2)	signal management /sigrelse(2)	sigset(2)
qdel(1) delete or	signal NQS requests	qdel(1)
fg_blank(3A) blank the output	signal of the frame grabber	fg_blank(3A)
sigcld(2I) modify SIGCLD on stop	signal option	sigcld(2I)
what to do on receipt of a	signal signal(2) specify	signal(2)
killpg(2B) send	signal to a process group	killpg(2B)
/fg_video_out(3A) select the video	signal types for I/O	fg_video_in(3A)
priority after handling a	signal /reset a process's	ucprelse(2I)
to a UCP priority on receipt of a	signal ucpsig(2I) set process	ucpsig(2I)
receipt of a signal	signal(2) specify what to do on	signal(2)
/sigrelse(2) sigignore(2)	sigpause(2) signal management	sigset(2)
sigpause(2)/ sigset(2) sighold(2)	sigrelse(2) sigignore(2)	sigset(2)
sigignore(2) sigpause(2) signal/	sigset(2) sighold(2) sigrelse(2)	sigset(2)
the frame grabber window	size fg_size(3A) determine	fg_size(3A)
get descriptor table	size getdtablesize(2B)	getdtablesize(2B)
getpagesize(2B) get system page	size	getpagesize(2B)
connection	sni_accept(3N) accept a	sni_accept(3N)
	sni_close(3N) close a connection	sni_close(3N)
server program	sni_connect(3N) connect to a	sni_connect(3N)
	sni_rxw(3N) receive a data buffer	sni_rxw(3N)
buffer	sni_txw(3N) transmit a data	sni_txw(3N)
accept a connection on a	socket accept(2B)	accept(2B)
bind(2B) bind a name to a	socket	bind(2B)
initiate a connection on a	socket connect(2B)	connect(2B)
listen for connections on a	socket listen(2B)	listen(2B)
getsockname(2B) get	socket name	getsockname(2B)
readv(2B) read input from a	socket	readv(2B)
receive a message from a	socket /recvfrom(2B) recvmsg(2B)	recv(2B)
sendmsg(2B) send a message from a	socket send(2B) sendto(2B)	send(2B)
writev(2B) write output to a	socket	writev(2B)
communication	socket(2B) create an endpoint for	socket(2B)
connected sockets	socketpair(2B) create a pair of	socketpair(2B)
get and set options on	sockets /setsockopt(2B)	getsockopt(2B)
create a pair of connected	sockets socketpair(2B)	socketpair(2B)
certnote.com(4) Intergraph	software certification/	certnote.com(4)
file fixes.com(4) Intergraph	software delivery documentation	fixes.com(4)
swap(2I) swap	space control	swap(2I)
memory efficient way vfork(2B)	spawn a new process in a virtual	vfork(2B)
csi_ccan(3A) cancel a	specific command on a CSI port	csi_ccan(3A)
system fmu_rcmd(3N) execute the	specified command on remote	fmu_rcmd(3N)
/flush the output for the	specified FORTRAN logical unit	flush(3F)
truncate a file to a	specified length ftruncate(2B)	ftruncate(2B)

a signal signal(2)	specify what to do on receipt of	signal(2)
lpq(1) BSD	spool queue examination program	lpq(1)
jobs from the BSD line printer	spooling queue lprm(1) remove	lprm(1)
generator random(3B)	srandom(3B) better random number	random(3B)
/request notification for a GPIB	SRQ condition	gpib_service(3A)
STANDCFG(4) optical disk	standalone configuration file	standcfg(4)
ansitape(4) ANSI	standard magtape labels	ansitape(4)
ksh(1) krsh(1) shell, the	standard/restricted command/	ksh(1)
standalone configuration file	STANDCFG(4) optical disk	standcfg(4)
fg_viw_start(3A) fg_viw_stop(3A)	start and stop video in a window	fg_viw_start(3A)
stat(5) data returned by_	stat system call	stat(5)
system call	stat(5) data returned by stat	stat(5)
statmon(4) record(4) recover(4)	state(4) status daemon directory/	statmon(4)
mailstats(1) display mail	statistics	mailstats(1)
state(4) status daemon directory/	statmon(4) record(4) recover(4)	statmon(4)
/record(4) recover(4) state(4)	status daemon directory and file/	statmon(4)
local/ ruptime(1) show host	status for each machine on the	ruptime(1)
csi_dstat_nw(3A) receive delayed	status from a CSI port	csi_dstat_nw(3A)
/receive unsolicited	status from a CSI port	csi_ustat(3A)
read the CSI port DR11	status lines csi_status(3A)	csi_status(3A)
lstat(2B) get file	status	lstat(2B)
qdev(1) display the	status of NQS devices	qdev(1)
qstat(1) display the	status of NQS queues	qstat(1)
csi_ucan(3A) cancel unsolicited	status requests on a CSI port	csi_ucan(3A)
sigcld(2I) modify SIGCLD on	stop signal option	sigcld(2I)
/fg_viw_stop(3A) start and	stop video in a window	fg_viw_start(3A)
supported batch limits and shell	strategy for the local host /show	qlimit(1)
/routines for returning a	stream to a remote command	rcmd(3B)
rexec(3B) return	stream to a remote command	rexec(3B)
bzero(3B) ffs(3B) bit and byte	string operations /bcmp(3B)	bstring(3B)
string(3B) index(3B) rindex(3B)	string operations	string(3B)
string operations	string(3B) index(3B) rindex(3B)	string(3B)
ffsinode(4)	structure of an FFS disk i-node	ffsinode(4)
the YP database and directory	structure ypfiles(4)	ypfiles(4)
terminal	stty(1) set the options for a	stty(1)
to NQS qpr(1)	submit a hardcopy print request	qpr(1)
qsub(1)	submit an NQS batch request	qsub(1)
dbm_clearerr(3B) database	subroutines /dbm_error(3B)	ndbm(3B)
strategy for the/ qlimit(1) show	supported batch limits and shell	qlimit(1)
swap(2I)	swap space control	swap(2I)
	swap(2I) swap space control	swap(2I)
dbg(1)	symbolic debugger	dbg(1)
sdb(1)	symbolic debugger	sdb(1)
readlink(2B) read the value of a	symbolic link	readlink(2B)
symlink(2B) make a	symbolic link to a file	symlink(2B)
to a file	symlink(2B) make a symbolic link	symlink(2B)
select(2B)	synchronous I/O multiplexing	select(2B)
intro(3A) introduction to the	synchronous/asynchronous I/O/	intro(3A)
(command interpreter) with C-like	syntax csh(1) a shell	csh(1)

identification number	sysid(2I) get the system hardware	sysid(2I)
readinfo(2I) read	system activity information	readinfo(2I)
CRM utility for monitoring	system activity topsys(1)	topsys(1)
backup(1) incremental file	system backup	backup(1)
stat(5) data returned by stat	system call	stat(5)
intro(2) introduction to	system calls and error numbers	intro(2)
/CRM utility for monitoring	system calls and faults	watcher(1)
types(5) primitive	system data types	types(5)
the specified command on remote	system fmu_rcmd(3N) execute	fmu_rcmd(3N)
receive files from a remote	system fmu_receive(3N)	fmu_receive(3N)
send files to a remote	system fmu_send(3N)	fmu_send(3N)
number sysid(2I) get the	system hardware identification	sysid(2I)
set or print name of current host	system hostname(1)	hostname(1)
mount(2) mount a file	system	mount(2)
directory contents on a remote	system /command that lists the	netls(1)
mount and unmount file	system npmount(1) npumount(1)	npmount(1)
to the optical disk file	system odintro(1) introduction	odintro(1)
getpagesize(2B) get	system page size	getpagesize(2B)
CRM utility for monitoring	system parameters monparam(1)	monparam(1)
restore(1) incremental file	system restore	restore(1)
fstab(4) file	system table	fstab(4)
mnttab(4) mounted file	system table	mnttab(4)
remotely mounted NFS file	system table rmtab(4)	rmtab(4)
ffsfs(4) format of file	system volume	ffsfs(4)
exports(4) NFS file	systems being exported	exports(4)
fstab(4) file system	table	fstab(4)
mnttab(4) mounted file system	table	mnttab(4)
remotely mounted NFS file system	table rmtab(4)	rmtab(4)
getdtablesize(2B) get descriptor	table size	getdtablesize(2B)
ypmapxlate(4) translation	table to handle long map names	ypmapxlate(4)
/fg_lut_out(3A) load the lookup	tables of a frame grabber	fg_lut_in(3A)
rtc(1) remote	tape control	rtc(1)
rtc_notify(3N) remote	tape control /rtc_deallocate(3N)	rtc_allocate(3N)
mt(1) magnetic	tape manipulation program	mt(1)
rtape(1) remote	tape manipulation program	rtape(1)
vmsbackup(1) read a VMS backup	tape	vmsbackup(1)
telnet(1) user interface to the	TELNET protocol	telnet(1)
TELNET protocol	telnet(1) user interface to the	telnet(1)
stty(1) set the options for a	terminal	stty(1)
/f77uninitio(3F) initialize or	terminate FORTRAN I/O from C	f77initio(3F)
exit(2) _exit(2)	terminate process	exit(2)
wait2(2I) wait for process to	terminate	wait2(2I)
wait3(2B) wait for process to	terminate	wait3(2B)
command	test(1) condition evaluation	test(1)
program	tftp(1) trivial file transfer	tftp(1)
merge(1)	three-way file merge	merge(1)
get/set value of interval	timer /setitimer(2B)	getitimer(2B)
floppy disk filters	to_flop(1) fr_flop(1) continuous	to_flop(1)
alt(1) Ada library	tool	alt(1)

determine the frame grabber	window size fg_size(3A)	fg_size(3A)
/plt_ctrl_nw(3A) send a control	word to the parallel port	plot_ctrl(3A)
/rplt_ctrl_nw(3A) send a control	word to the ROP parallel port	plot_ctrl_rop(3A)
rcsclean(1) clean up	working files	rcsclean(1)
pdi_write(3A) pdi_write_nw(3A)	write data from memory to a PDI/	pdi_write(3A)
gpib_write(3A) gpib_write_nw(3A)	write data to a GPIB device	gpib_write(3A)
aux_write(3A) aux_write_nw(3A)	write data to a serial port	aux_write(3A)
/fpe_write_dma_nw(3A)	write data to an FPE coprocessor	fpe_write_dma(3A)
/plt_data(3A) plt_data_nw(3A)	write data to the parallel port	plot_data(3A)
/rplt_data(3A) rplt_data_nw(3A)	write data to the ROP parallel/	plot_data_rop(3A)
writev(2B)	write output to a socket	writev(2B)
fpe_cancel_dma(3A) cancel	write request to an FPE/	fpe_cancel_dma(3A)
write(2)	write to a file	write(2)
	write(2) write to a file	write(2)
socket	writev(2B) write output to a	writev(2B)
allocate/deallocate an event/	xio_allocef(3A) xio_deallocef(3A)	xio_allocef(3A)
flag mask/ xio_readef(3A)	xio_clref(3A) xio_setef(3A) event	xio_readef(3A)
xio_allocef(3A)	xio_deallocef(3A)/	xio_allocef(3A)
of an asynchronous request/	xio_notify(3A) notify a process	xio_notify(3A)
xio_setef(3A) event flag mask/	xio_readef(3A) xio_clref(3A)	xio_readef(3A)
xio_readef(3A) xio_clref(3A)	xio_setef(3A) event flag mask/	xio_readef(3A)
xio_wflor(3A) asynchronous event/	xio_waitfr(3A) xio_wfland(3A)	xio_waitfr(3A)
asynchronous/ xio_waitfr(3A)	xio_wfland(3A) xio_wflor(3A)	xio_waitfr(3A)
xio_waitfr(3A) xio_wfland(3A)	xio_wflor(3A) asynchronous event/	xio_waitfr(3A)
to console devices on the local	XNS network /send a message	netmsg(1)
server.dat(4)	XNS server information file	server.dat(4)
yperr_string(3R) ypprot_err(3R)	YP client interfac /yp_master(3R)	ypclnt(3R)
structure ypfiles(4) the	YP database and directory	ypfiles(4)
ypcat(1) print values in a	YP database	ypcat(1)
set or display name of current	YP domain domname(1)	domname(1)
value of one or more keys from a	YP map ypmatch(1) print the	ypmatch(1)
change login password in	YP yppasswd(1)	yppasswd(1)
update user password in	YP yppasswd(3R)	yppasswd(3R)
/yp_first(3R) yp_next(3R)	yp_all(3R) yp_order(3R)/	ypclnt(3R)
/yp_get_default_domain(3R)	yp_bind(3R) yp_unbind(3R)/	ypclnt(3R)
database	ypcat(1) print values in a YP	ypcat(1)
yp_get_default_domain(3R)/	ypclnt(3R)	ypclnt(3R)
YP/ /yp_order(3R) yp_master(3R)	yperr_string(3R) ypprot_err(3R)	ypclnt(3R)
directory structure	ypfiles(4) the YP database and	ypfiles(4)
/yp_unbind(3R) yp_match(3R)	yp_first(3R) yp_next(3R)/	ypclnt(3R)
yp_bind(3R)/ ypclnt(3R)	yp_get_default_domain(3R)	ypclnt(3R)
to handle long map names	ypmapxlate(4) translation table	ypmapxlate(4)
/yp_all(3R) yp_order(3R)	yp_master(3R) yperr_string(3R)/	ypclnt(3R)
or more keys from a YP map	ypmatch(1) print the value of one	ypmatch(1)
/yp_bind(3R) yp_unbind(3R)	yp_match(3R) yp_first(3R)/	ypclnt(3R)
/yp_match(3R) yp_first(3R)	yp_next(3R) yp_all(3R)/	ypclnt(3R)
/yp_next(3R) yp_all(3R)	yp_order(3R) yp_master(3R)/	ypclnt(3R)
in YP	yppasswd(1) change login password	yppasswd(1)
in YP	yppasswd(3R) update user password	yppasswd(3R)

Commands (1)

NAME

intro - introduction to commands and application programs

DESCRIPTION

This section describes commands available for the CLIX System. A portion of the commands is standard System V commands that have been modified under CLIX. The remainder are CLIX-specific commands.

Manual Page Command Syntax

Unless otherwise noted, commands described in the **SYNOPSIS** section of a manual page accept options and other arguments according to the following syntax and should be interpreted as explained below.

name [-*option* ...] [*cmdarg* ...]

where:

[ ]	Surround an *option* or *cmdarg* that is not required.
...	Indicates multiple occurrences of the *option* or *cmdarg*.
name	The name of an executable file.
option	(Always preceded by a "-".) *noargletter* ... or *argletter optarg* [,...]
noargletter	A letter representing an option without an option-argument. More than one *noargletter* option can be grouped after one "-" (rule 5, below).
argletter	A letter representing an option requiring an option-argument.
optarg	An option-argument (character string) satisfying a preceding *argletter*. Note that groups of *optargs* following an *argletter* must be separated by commas or separated by white space and quoted (rule 8, below).
cmdarg	Path name (or other command argument) not beginning with "-", or "-" by itself indicating the standard input.

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new commands will obey them. *getopts*(1) should be used by all shell procedures to parse positional parameters and to check for legal options. It supports rules 3-10 below. The command must enforce the other rules.

1. Command names (*name* above) must be between two and nine characters long.
2. Command names must include only lowercase letters and digits.
3. Option names (*option* above) must be one character long.
4. All options must be preceded by "-".
5. Options with no arguments may be grouped after a single "-".
6. The first option-argument (*optarg* above) following an option must be preceded by white space.

7. Option-arguments cannot be optional.
8. Groups of option-arguments following an option must either be separated by commas or separated by white space and quoted (e.g., *-o xxx,z,yy* or *-o "xxx z yy"*).
9. All options must precede operands (*cmdarg* above) on the command line.
10. "--" may be used to indicate the end of the options.
11. The order of the options relative to one another should not matter.
12. The relative order of the operands (*cmdarg* above) may affect their significance in ways determined by the command with which they appear.
13. "-" preceded and followed by white space should only be used to mean standard input.

SEE ALSO

exit(2) in the *CLIX System V Programmer's & User's Reference Manual.*
getopts(1) in the *UNIX System V User's Reference Manual.*
wait(2), getopt(3C) in the *UNIX System V Programmer's Reference Manual.*
"How to Get Started" at the front of *UNIX System V User's Reference Manual.*

DIAGNOSTICS

At termination, each command returns two bytes of status—one supplied by the system that gives the cause for termination, and (in the case of "normal" termination) one supplied by the program (see *wait*(2) and *exit*(2)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and nonzero to indicate troubles such as erroneous parameters or bad or inaccessible data. The latter byte is called "exit code", "exit status", or "return code", and is described only where special conventions are involved.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused when encountering a null character (the string terminator) within a line.

NAME

ab - Ada program beautifier

SYNOPSIS

ab [-**i** *indent*] [-**l** *linelen*] [-**w**] [-**K** [**lus**]] [-**I** [**lus**]] [*file* ...]

DESCRIPTION

ab reads Ada source programs either from the specified *file*s or from standard input and writes them to standard output with spacing and indentation that displays the structure of the code. If the Ada source has syntax errors, *ab* displays the line number and token of the first error before exiting. It may be used as an Ada syntax checker.

The following options are available:

-i *indent*	Set the indentation to *indent*, which must be a natural number. The default indentation is four spaces.
-l *linelen*	Set the line length to *linelen*, which must be a positive number. The default line length is 64. Warnings will be generated if this is exceeded. Note that *ab* will not always prevent line overflow.
-w	Suppress warning messages.
-K [**lus**]	Indicate the format of Ada keywords. The **l** option puts them in lowercase, the **u** option puts them in uppercase, and the **s** option leaves the case unchanged. The **s** option is used by default.
-I [**lus**]	Indicate the format of Ada identifiers. The **l** option puts them in lowercase, the **u** option puts them in uppercase, and the **s** option leaves the case unchanged. The **s** option is used by default.

SEE ALSO

act(1), Ac(1).

NAME

Ac - Ada compiler

SYNOPSIS

Ac [*option* ...] *file* ...

DESCRIPTION

Ac is the York (Release 4) Ada compiler. Arguments whose names end with **.H** and **.A** are interpreted as Ada source files; they are compiled, and each object program whose name is that of the source with **.O** substituted for **.H** and **.o** for **.A** remains in the file. In the same way, arguments whose names end with **.s** are assembled and a **.o** file is produced. Arguments whose names end with **.c** are interpreted as C source files; they are compiled and placed in a **.o** file to be loaded if required. In addition, **.o** and **.a** files may be passed as arguments to be loaded along with other **.o** files. Note that any Ada programs with foreign language bodies must have the corresponding **.c** or **.o** file explicitly mentioned on the command line. These and any other file name arguments are passed to *ld*(1) and are loaded together to form an executable *a.out*(4) file.

In addition to creating an object file, a compilation updates a library file, **ADA-LIBRARY**, for each compilation unit in the source file. (The library file is created if it does not already exist.)

The library file contains the locations of the units in the program library and is read by subsequent compilations to enforce the separate compilation rules of the language.

Ac recognizes the following options:

-M *identifier*	Make the compilation unit *identifier* the main subprogram. The main subprogram must be a procedure body with no parameters defined. **-M main** is assumed if this option is not specified.
-v	Produce verbose error messages. In particular, the compiler will attempt to isolate faults within expressions detected during overload resolution.
-w	Suppress all warning messages from the Ada compiler.
-c	Do not load the resulting **.o** files together.
-S	Save the assembly code output of the compiler in a file with suffix **.s** or **.S**. No object files will be produced.
-o *outfile*	Name the resulting file *outfile* rather than the default **a.out**.
-O	Invoke the C object code improver on the compiler output.
-L	Load only functions that are actually called (directly or indirectly). This can considerably decrease the size of the executable at the expense of a longer assembly and load time. This optimization does not apply to functions

entirely local to a unit. This option is incompatible with the **-g** option. When this option is given, the **-Re** option is automatically switched on. Otherwise, few savings accrue. All units that make up an executable file must be compiled with the same setting of this flag. The standard Ada library provided is compiled with this option.

-G Load a global garbage collector to replace the usual garbage collection scheme. The latter does not reclaim heap space for access types defined within a library package.

-p Generate profiling information.

-g Generate extra symbol table information for *dbg*(1). Until the sources become available to the compiler developers, *dbg*(1) is very unreliable when used with Ada programs. The user can set break points, but printing the values of variables does not always give the expected results. It can cause the debugger to crash.

-V Print the version number of the compiler.

-I *directory* Search for library files in the named *directory*, in the directory of the source file, and in the standard library **/usr/lib/Ada**. Any number of **-I** options may be given. Directories are searched in the following order: directory of the source file, directories specified by **-I** options (in the order given), **/usr/lib/Ada**.

-l*x* Specify a library name of the form **/lib/lib***x***.a**, where *x* is a string. If the library does not exist, *ld*(1) tries **/usr/lib/lib***x***.a**. A library is searched when its name is encountered, so the placement of a **-l** is significant.

-R*string* Suppress the following run time checks according to the characters in the given *string*. (Compare pragma suppress.)

a	access_check
d	discriminant_check
i	index_check
l	length_check
r	range_check
z	division_check
o	overflow_check
e	elaboration_check
s	storage_check
x	all of the above checks are suppressed

All other flags are passed to the loader.

FILES

libada.a	runtime support library for basic features

libtask.a	runtime support library for tasking feature
libadastand.a	basic I/O and memory management library
/bin/ld	link editor (standard AT&T)
/bin/as	assembler
/lib/crt0.o	runtime startup code
/usr/lib/Ada/ald	Ada-specific link editor
/usr/lib/Ada/asplit	Ada-specific assembler
/usr/lib/Ada/elab_clipper	entry label generation program
/usr/lib/Ada/ac_clipper	Ada CLIPPER compiler

SEE ALSO

adep(1).

NAME

acctcom - search and print process accounting files

SYNOPSIS

acctcom [*option* ...] [*file* ...]

DESCRIPTION

acctcom reads *file*, standard input, or **/usr/adm/pacct** in the form described by *acct*(4) and writes selected records to standard output. Each record represents the execution of one process. The output shows the COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SEC), CPU (SEC), MEAN SIZE (K), F (the *fork*(2)/*exec*(2) flag: 1 for *fork*(2) without *exec*(2)), STAT (the system exit status), HOG FACTOR, KCORE MIN, CPU FACTOR, CHARS TRNSFD, and BLOCKS READ (total blocks read and written).

A **#** is prepended to the command name if the command was executed with super-user privileges. If a process is not associated with a known terminal, a ? is printed in the TTYNAME field.

If no *file*s are specified and if standard input is associated with a terminal or **/dev/null** (as is the case when using **&** in the shell), **/usr/adm/pacct** is read; otherwise, standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward (in chronological order by process completion time). The file **/usr/adm/pacct** is usually the current file to be examined; a busy system may need several such files of which all but the current file are found in **/usr/adm/pacct***?*. The *option*s are as follows:

-a	Show average statistics about the processes selected. The statistics will be printed after the output records.
-b	Read backwards showing latest commands first. This option has no effect when standard input is read.
-f	Print the *fork*(2)/*exec*(2) flag and system exit status columns. The numeric output for this option will be in octal.
-h	Instead of showing mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This hog factor is computed as follows: (total CPU time) / (elapsed time)
-i	Print columns containing the I/O counts.
-k	Instead of showing memory size, show total kcore-minutes.
-m	Show mean core size (the default).
-r	Show CPU factor (user time / (system time + user time)).
-t	Show separate system and user CPU times.
-v	Exclude column headings from the output.

-l *line*	Show only processes belonging to terminal **/dev/***line*.
-u *user*	Show only processes, belonging to *user*, specified by: a user ID, a login name that is then converted to a user ID, a **#**, which designates only those processes executed with super-user privileges, or **?**, which designates only processes associated with unknown user IDs.
-g *group*	Show only processes belonging to *group*. *Group* may be designated by either the group ID or group name.
-s *time*	Select processes existing at or after *time*, given in the format *hr* [:*min* [:*sec*]].
-e *time*	Select processes existing at or before *time*.
-S *time*	Select processes starting at or after *time*.
-E *time*	Select processes ending at or before *time*. Using the same *time* for both **-S** and **-E** shows the processes that existed at *time*.
-n *pattern*	Show only commands matching *pattern* that may be a regular expression as in *ed*(1) except that + means one or more occurrences.
-q	Do not print any output records; print only the average statistics as with the **-a** option.
-o *ofile*	Copy selected process records in the input data format to *ofile*; suppress standard output printing.
-H *factor*	Show only processes that exceed *factor*, which is the hog factor as explained in option **-h** above.
-O *sec*	Show only processes with CPU system time exceeding *sec* seconds.
-C *sec*	Show only processes with total CPU time, system plus user, exceeding *sec* seconds.
-I *chars*	Show only processes transferring more characters than the cutoff number given by *chars*.

FILES

/etc/passwd	used for login name to user ID conversions
/usr/adm/pacct	current process accounting file
/etc/group	group ID information

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M) in the *CLIX System Administrator's Reference Manual*.
acct(2), acct(4), utmp(4) in the *UNIX System V Programmer's Reference Manual*.

BUGS

acctcom reports only on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time, *time* is interpreted as occurring on the previous day.

NAME

adb - absolute debugger

SYNOPSIS

adb [**-w**] [*objfil* [*corfil*]]

DESCRIPTION

adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of CLIX system programs.

Objfil is normally an executable program file, preferably containing a symbol table. If it does not contain a symbol table, the symbolic features of *adb* cannot be used, although the file can still be examined. The default for *objfil* is **a.out**. *Corfil* is assumed to be a *core*(4) image file produced after executing *objfil*. The default for *corfil* is **core**.

Requests to *adb* are read from the standard input and responses are to the standard output. If the **-w** option is present, *objfil* and *corfil* are created, if necessary, and opened for reading and writing so that files can be modified using *adb*. *adb* ignores QUIT. INTERRUPT causes return to the next *adb* command.

In general, requests to *adb* have the form

[*address*][**,***count*][*command*][**;**]

If *address* is present, *dot* is set to *address*. Initially, *dot* is set to 0. For most commands, *count* specifies how many times the command will be executed. The default *count* is 1. *Address* and *count* are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged, addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see **Addresses.**

Expressions

.	The value of *dot*.
+	The value of *dot* incremented by the current increment.
^	The value of *dot* decremented by the current increment.
"	The last *address* typed.
integer	A hexadecimal, decimal, or octal number, depending on whether the number begins with **0x**, **0t**, or **0o**, respectively. Otherwise, a hexadecimal number.
*integer***.***fraction*	A 32-bit, floating-point number.
'*cccc***'**	The ASCII value of up to four characters. A \ may be used to escape a **'**.

<*name*	The value of *name* is a variable name or a register name. *adb* maintains a number of variables (see **Variables**) named by single letters or digits. If *name* is a register name, the register value is obtained from the subprocess or the system header in *corfil*. The register names are **r0** to **r15**, **f0** to **f7**, **f0x** to **f7x**, **psw**, **ssw**, and **pc** for the general, floating-point, and processor registers. **Fp** and **sp** are synonyms for **r14** and **r15**, respectively. **F0** to **f7** are the low-order 32 bits of the floating-point registers and **f0x** to **f7x** are the high-order 32 bits.
symbol	A *symbol* is a sequence of uppercase or lowercase letters, underscores, or digits not starting with a digit. \ may be used to escape other characters. The value of the *symbol* is taken from the symbol table in *objfil*. An initial _ or ~ will be prefixed to *symbol* if needed.
_*symbol*	In C, the "true name" of an external symbol begins with _. It may be necessary to utter this name to distinguish it from internal or hidden variables of a program.
(*exp*)	The value of the expression *exp*.

Monadic operators:

**exp*	The contents of the location addressed by *exp* in *corfil*.
@*exp*	The contents of the location addressed by *exp* in *objfil*.
−*exp*	Integer negation.
~*exp*	Bitwise complement.

Dyadic operators are left associative and are less binding than monadic operators.

e1+*e2*	Integer addition.
e1−*e2*	Integer subtraction.
*e1***e2*	Integer multiplication.
e1%*e2*	Integer division.
e1&*e2*	Bitwise conjunction.
e1|*e2*	Bitwise disjunction.
e1#*e2*	*E1* rounded up to the next multiple of *e2*.

Commands

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands ? and / may be followed by * (see **Addresses**).)

?*f* Locations starting at *address* in *objfil* are printed according to the format *f*. *Dot* is incremented by the sum of the increments for each format letter.

/f	Locations starting at *address* in *corfil* are printed according to the format *f*, and *dot* is incremented as it is for **?**.
=f	The value of *address* is printed in the styles indicated by the format *f*. (For **i** format, **?** is printed for the parts of the instruction that reference subsequent words.)

A format consists of one or more characters that specify a printing style. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format, *dot* is incremented by the amount given for each format letter. If no format is given, the last format is used. The format letters available are as follows:

o 2	Print 2 bytes in octal. All octal numbers output by *adb* are preceded with **0**.
O 4	Print 4 bytes in octal.
q 2	Print in signed octal.
Q 4	Print in long signed octal.
d 2	Print in decimal.
D 4	Print in long decimal.
x 2	Print 2 bytes in hexadecimal.
X 4	Print bytes in hexadecimal.
u 2	Print as an unsigned decimal number.
U 4	Print long unsigned decimal.
f 4	Print the 32-bit value as a floating-point number.
F 8	Print double floating-point.
b 1	Print the addressed byte in octal.
c 1	Print the addressed character.
C 1	Print the addressed character using the following escape convention. Character values 000 to 040 are printed as **@** followed by the corresponding character in the range 0100 to 0140. The character **@** is printed as **@@**.
s *n*	Print the addressed characters until a zero character is reached. The value *n* is the length of the string including its zero terminator.
S *n*	Print a string using the **@** escape convention. The value *n* is the length of the string including its zero terminator.
Y 4	Print 4 bytes in date format.
i 2	Print as CLIPPER™ instructions.
a 0	Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

/	Local or global data symbol.
?	Local or global text symbol.
=	Local or global absolute symbol.

p 2	Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.
t 0	When preceded by an integer, tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.

r 0	Print a space.
n 0	Print a newline.
"..." 0	Print the enclosed string.
^	*Dot* is decremented by the current increment. Nothing is printed.
+	*Dot* is incremented by 1. Nothing is printed.
−	*Dot* is decremented by 1. Nothing is printed.

newline

Repeat the previous command with a *count* of 1.

[?/]l *value mask*

Words starting at *dot* are masked with *mask* and compared to *value* until a match is found. The *mask* argument is optional. The mask used is 0xff for **b** and **o**, 0xffff for **w**, -1 for **l**, *mask* if supplied, or -1 by default. The *incr* argument is optional. *Dot* is incremented by 1 for **b**, 2 for **w**, 4 for **l**, the size of the instruction for **o**, *incr* if specified, or 1 by default. If a match is found, *dot* is set to the matched location. Otherwise, *dot* is unchanged.

[?/]**w** *value ...*

Write the 2-byte *value* into the addressed location. If the command is **W**, write 4 bytes. Odd addresses are allowed under the CLIX system when writing to the subprocess address space.

[?/]**m** *b1 e1 f1*[?/]

New values for (*b1*, *e1*, *f1*) are recorded. If less than three expressions are given, the remaining map parameters are unchanged. If the ? or / is followed by *, the second segment (*b2*, *e2*, *f2*) of the mapping is changed. If the list is terminated by ? or /, the file (*objfil* or *corfil*, respectively) is used for subsequent requests. (So that, for example, /**m**? will cause / to refer to *objfil*.)

>*name*

Dot is assigned to the variable or register named.

! A shell is called to read the remainder of the line following **!**.

$*modifier*

Miscellaneous commands. The available *modifiers* are as follows:

<*f*	Read commands from the file *f* and return.
>*f*	Send output to the file *f*, which is created if it does not exist.
r	Print the general registers and the instruction addressed by the PC. *Dot* is set to the PC. All registers are printed as if they were integer registers, including the floating-point registers.
b	Print all breakpoints and their associated counts and commands.
c	C stack backtrace. Routine names are printed for routines that set up a frame pointer (see *cc*(1)). If *count* is given, only

the first *count* frames are printed.

e	The names and values of external variables are printed.
w	Set the page width for output to *address* (default 80).
s	Set the limit for symbol matches to *address* (default 255).
o	All integers input are regarded as octal.
d	Reset integer input as described in *Expressions*.
q	Exit from *adb*.
v	Print all nonzero variables in octal.
m	Print the address map.

:*modifier*

Manage a subprocess. Some process management commands do not work until the process is created. **:r**, and **:s**, create a process first, if necessary. Available modifiers are as follows:

b*c*	Set breakpoint at *address*. The breakpoint is executed *count*−1 times before causing a stop. Each time the breakpoint is encountered, the command *c* is executed. If this command sets *dot* to zero, the breakpoint causes a stop.
d	Delete breakpoint at *address*.
r	Run *objfil* as a subprocess. If *address* is given explicitly, the program is entered at this point. Otherwise, the program is entered at its standard entry point. The value *count* specifies how many breakpoints are ignored before stopping. Arguments to the subprocess may be supplied on the line the command is on. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on when the subprocess is entered.
c*s*	The subprocess is continued with signal *s* (see *signal*(2)). If *address* is given, the subprocess continues at this address. If no signal is specified, the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.
s*s*	Same as for **c** except that the subprocess is single stepped *count* times. If there is no current subprocess, *objfil* is run as a subprocess as it is for **r**. In this case, no signal can be sent; the remainder of the line is treated as arguments to the subprocess.
k	The current subprocess, if any, is terminated.

Variables

Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows.

0	The last value printed.

On entry, the following are set from the system header in the *corfil*. If *corfil* does not appear to be a *core*(4) file, these values are set from *objfil*.

b	The base address of the data segment.
d	The data segment size.
e	The entry point.
m	The "magic" number (0405, 0407, 0410 or 0411).
s	The stack segment size.
t	The text segment size.

Addresses

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

$$b1 \leqslant address < e1 \rightarrow file\ address = address + f1 - b1$$

Otherwise, it is calculated as follows:

$$b2 \leqslant address < e2 \rightarrow file\ address = address + f2 - b2$$

If one of these methods does not succeed, the requested *address* is not legal. In some cases (i.e., for programs with separated I and D space), the two segments for a file may overlap. If a ? or / is followed by an *, only the second triple is used.

The initial setting of both mappings is suitable for normal *a.out*(4) and *core*(4) files. If neither file is the kind expected, for that file *b1* is set to 0, *e1* is set to the maximum file size, and *f1* is set to 0. In this way, the whole file can be examined with no address translation.

For *adb* to be used on large files, all appropriate values are kept as signed, 32-bit integers.

FILES

/dev/mem
/dev/swap
a.out
core

SEE ALSO

a.out(4), core(4).
ptrace(2) in the *UNIX System V Programmer's Reference Manual*.

DIAGNOSTICS

The string "adb" is displayed when there is no current command or format. Otherwise, comments about inaccessible files, syntax errors, abnormal command termination, etc. are displayed. Exit status is 0 unless the last command failed or returned a nonzero status.

BUGS

A breakpoint set at the entry point is not effective on initial entry to the program.

Local variables whose names are the same as an external variable may cause problems in accessing the external.

NAME

adep - Ada program makefile generator

SYNOPSIS

adep [**-f** *makefile*] [**-M** *main-unit*] [**-o** *objfile*] [**-t** *target*] [**-c** *compiler*] [**-lx**] *file* ... [**-I** *file* ...]

DESCRIPTION

adep takes a set of files and constructs a *makefile* containing all dependencies necessary to produce an executable *a.out*(4) file. The user may request a rule for an executable file to be produced or a rule to maintain only the object files. A rule for cleaning up the directory is also available (invoked with the **make clean** command.)

If the name of the *makefile* is not supplied, **makefile** is assumed.

The following options are available:

-f *makefile*	Name a *makefile* explicitly.
-M *main-unit*	Cause *adep* to treat *main-unit* as the name of the main program unit. **main** is used by default.
-o *objfile*	Specify the name of the executable file to be produced. By default, **a.out** is produced.
-t *target*	Indicate that no executable file will be produced and specify the name of the main target.
-c *compiler*	Specify the Ada compiler to be used. By default, *Ac*(1) is assumed.
-I *file* ...	Indicate that all following Ada source files should be included as possible prerequisites but not as targets in the *makefile*. Typically this option could be used to construct dependencies between directories or program libraries. (This option may follow the list of files.)
-lx	Specify a library name of the form **/lib/libx.a**, where *x* is a string. If the library does not exist, **/usr/lib/libx.a** is tried. Libraries specified in this manner are assumed to contain object files to be loaded to create the required executable.

Extra flags to the Ada compiler may be set as follows :

```
        adep "AFLAGS=-O -v" *.[HA]
```

This passes the optimizer flag and verbose flag to all calls of the Ada compiler. In a similar way, the user may pass any variable and value to *adep* (such as CFLAGS) and these will be included in the *makefile*.

File suffixes are interpreted as follows:

A file with the suffix **.H** or **.A** is an Ada source file and will be scanned to determine the list of dependencies.

A file with the suffix **.c** is a C source file and will be compiled and loaded with other object files to produce any required executable.

A file with the suffix **.s** is an assembly language source file and will be assembled and loaded with other object files to produce any required executable.

A file with the suffix **.o** is an object file and will be loaded with other object files to produce any required executable. Note that object files produced from Ada source files are loaded automatically and must not be included here.

A file with the suffix **.a** is an archive file containing object files and will be loaded with other object files to produce any required executable. Alternatively, the user may specify archive files using the **-l** option (see *cc*(1) and *ld*(1)).

All other files are ignored.

adep marks the start of the generated dependencies in the *makefile* with the following line:

```
# Ada dependencies generated by adep
```

adep marks the end of them with the following line:

```
# End of Ada dependencies
```

Everything outside of these lines remains untouched by *adep*, so any additions to the *makefile* by the user will not be affected by later use of *adep* with this file.

The dependency set is replaced each time *adep* is used; dependencies cannot be added or removed incrementally.

adep is not an Ada language parser. It looks for **with** clauses, **separate** clauses, and the unit or subunit information. It then attempts to reach the end of the current unit before repeating the process for any following units in the file. It does so by relying on the positioning of semicolons and certain reserved words (such as **end**, **loop**, and **begin**.) Consequently, a serious syntax error will cause *adep* to terminate with an error message and a line number. In this case running the Ada compiler may help to obtain more information about the error.

SEE ALSO

Ac(1).
make(1) in the *UNIX System V Programmer's Reference Manual.*

BUGS

For an instantiation of an external generic, *adep* records a dependency on the generic body and all subunits of the generic. *adep* will become confused if different generics have the same name or a generic has the same name as the unit in which it occurs. However, it will usually print a message in this case.

NAME

aligntrap - set/report alignment trap disposition per executable

SYNOPSIS

aligntrap [-**y** | -**n**] *file* ...

DESCRIPTION

C200 and C300 revisions of the CLIPPER processor will optionally trap and provide a signal to a process on misaligned memory accesses. The disposition of alignment traps is controlled per executable through a flag in the CLIX system header of the common object (COFF) file. This flag is read by the operating system during the *exec*(2) system call and determines the alignment trap action for the created process. *aligntrap* with no option will report the current state of the alignment trap enable flag for each specified *file*. Specifying -**y** enables the alignment trap for processes created from *file*. Specifying -**n** disables the alignment trap for processes created from *file*.

SEE ALSO

a.out(4) in the *CLIX Programmer's & User's Reference Manual.*

NOTES

Checks are performed to verify that each file operated on is actually a CLIPPER executable.

NAME

alt - Ada library tool

SYNOPSIS

alt [*file*]

DESCRIPTION

alt allows the interactive display and editing of an Ada library file produced by *Ac*(1). If the file is a directory, the file **ADA-LIBRARY** is sought within the directory. If no file is specified, **ADA-LIBRARY** is assumed.

alt recognizes the following commands:

l [**-dlst**] [*name* ...] List the library contents. The default format lists each unit name followed by a description of the unit kind.

The **l** command recognizes the following options:

-d Display the dependencies recorded for each compilation unit.

-l List in long format giving the source file and last compilation date of each compilation unit.

-s Display subunit information. The full parent name is given for each subunit and a list of all subunits is given for each parent.

-t Sort compilation units by compilation times (latest first) instead of by name.

d [**-i**] *name* ... Delete all named compilation units from the library. If the **-i** (interactive) option is specified, the user will be asked whether each compilation unit should be deleted.

w [*file*] Save the library in the given file. If no file is specified, the original is used.

q
end-of-file Quit the session.

!*command* Escape to the shell to execute *command*.

Names given in an **l** or **d** command can be expressed as regular expressions in the style of *egrep*(1).

SEE ALSO

Ac(1), adep(1).
egrep(1) in the *UNIX System V User's Reference Manual.*

NAME

ansitape - ANSI-standard magtape label program

SYNOPSIS

ansitape [-] **t** | **x** | **r** | **c** [**vqfaei3**] [**mt**=*device*] [**vo**=*volume-name*] [**rs**=[**r** | *record-size*]] [**bs**=*block-size*] [**rf**=[**v** | **f**]] [**cc**=[**i** | **f** | **e**]] *file-name* ...

DESCRIPTION

ansitape reads, writes, and creates magtapes conforming to the American National Standards Institute (ANSI) standard for magtape labeling. Primarily, this is useful to exchange tapes with VAX/VMS™ which makes this kind of tape by default.

ansitape is controlled by a function key letter (t, **x**, c, or **r**). Various options modify the format of the output tape.

The function letters describe the overall operation desired. A minus sign (-) is allowed, but optional, to introduce the first keyword option set. The function is specified with one of the following:

r Write to a magtape.

c Create a new magtape. The tape is initialized with a new ANSI volume header. All files previously on the tape are destroyed. This option implies **r**.

x Extract all files from the tape. Files are placed in the current directory. Protection is read/write to everyone, modified by the current *umask*(2).

t List all file names on the tape.

These key letters may follow the function letter:

v Normally, *ansitape* works silently; the **v** (verbose) option displays the name of each file *ansitape* treats, preceded by the function letter. It also displays the volume name of each tape as it is mounted. When used with the **t** option, *ansitape* displays the number of tape blocks used by each file, the record format, and the carriage control option.

q Query before writing. On write (**c** or **r** options), this causes *ansitape* to ask before writing to the tape. On extract operations, *ansitape* displays the CLIX path name and asks if it should extract the file. Any response starting with a "y" or "Y" means yes, and any other response (including an empty line) means no.

f File I/O is done to standard I/O instead. For example, when writing a tape file that will contain a lint listing, the follwing could be specified:

```
lint xyz.c | ansitape rf xyz.lint
```

instead of

```
lint xyz.c > /tmp/xyz.lint
ansitape r /tmp/xyz.lint
rm /tmp/xyz.lint
```

When reading, this option causes the extracted files to be sent to **stdout** instead of to a disk file.

a The tape should be read or written with the ASCII character set. This is the default.

e The tape should be written with the EBCDIC character set. The mapping is the same one used by the *dd*(1M) program with **conv=ebcdic**. This option is automatically enabled if IBM® format labels are selected.

i Use IBM format tape labels. The IBM format is similar but not identical to the ANSI standard. The major difference is that the tape will contain no HDR3 or HDR4 records and restricts the names of the files on the tape to 17 characters. This option automatically selects the EBCDIC character set for output. To make an IBM format label on a tape using the ASCII character set, use the option sequence **ia**.

3 Do not write HDR3 or HDR4 labels. The HDR3 label is reserved for the use of the operating system that created the file. HDR4 is for overflow of file names longer than the 17 characters allocated in the HDR1 label. Not all systems process or ignore these labels correctly. This switch suppresses the HDR3 and HDR4 labels when the tape will be transferred to a system that cannot support these types of labels.

Function modifiers should be given as a separate argument to *ansitape*. Multiple modifiers may be specified. They must appear after the key-letter options above and before any file name arguments.

mt=*device*
Select an alternate drive on which the tape is mounted. The default is **/dev/rmt/mt0n**.

vo=*volume-name*
Specify the name of the output volume. Normally, this defaults to the first six characters of the login name. The string "UNIX™" is used as the default if *ansitape* cannot determine the login name.

rs=*record-size*
Specify the output record size in bytes. This is the maximum size in the case of variable-format files. This option also turns on the fixed-record-format option. Thus, for variable record sizes with a smaller maximum,

rs=*record-size* **rf=v**

must be specified. When the record size is manually given, *ansitape* does not read disk files to determine the maximum record length.

rs=r This is a variant of the **rs=** option. It causes *ansitape* to read all disk files for record size, regardless of their size. Normally, files larger than 100K bytes are not scanned for record size. Using this option also implies variable-length records.

bs=*block-size*
Specify the output block size, in bytes. As many records as will fit are written into each physical tape block. ANSI standards limit this to 2048 bytes (the default), but more or less may be specified. Specifying more may prevent some systems from reading the tape.

rf=v Record format is variable-length. In other words, they are text files. This is the default and normally should not be changed.

rf=f Record format is fixed-length. This is usually a bad choice, and should be reserved for binary files.

cc=i Carriage control is implied (default). Unlike CLIX text files where records are delimited by a newline character, ANSI files do not normally include the newline as part of the record. Instead, a newline is automatically added to the record whenever the record is sent to a printing device.

cc=f Carriage control FORTRAN. Each line is expected to start with a FORTRAN carriage-control character. *ansitape* does not insert these characters automatically, it merely marks the file as having them. This is of limited usefulness.

cc=e Carriage control is embedded. Carriage control characters (if any) are part of the data records. This is usually used with binary data files.

Writing ANSI Tapes

The list of files on the command line is written to the tape. A full CLIX path name may be specified. However, only the last path name component (everything after the last /) is used as the file name on the tape.

Normally, regular text files are to be exchanged. *ansitape* reads the files one line at a time and transfers them to the tape. The newline character at the end of each line is removed, and the file is written in a variable-length record format. Variable-format files have the length of the longest record specified in a file header. Therefore, *ansitape* will read each input file from disk before the file is written to tape, to determine the maximum record size. The read is skipped if the file is more than 100,000 bytes long. The default carriage control (implied) instructs the other host to restore the newline character before printing the record.

If *ansitape* assumes that the input file is a CLIX text file (FORTRAN or implied carriage control), it will automatically strip the CLIX newline from the end of each record. Stripping is not done with embedded carriage control (**cc=e**) files. If the size of a nontext file (**cc=e**) is not a multiple of the record size, the partial record at the end will be lost.

For binary files, fixed-length records should be used. VAX/VMS normally uses a record length of 512 bytes for things like directories and executable files, but data files may have any record length. Binary files should be flagged for embedded carriage control.

Reading ANSI Tapes

When reading, the input file list is presumed to be the names of files to be extracted from the tape. The shell wildcard characters asterisk (*) and question mark (?) may be used. Of course, they must be quoted to prevent the shell from interpreting them before *ansitape* sees them.

None of the options for record format or carriage control need to be specified when reading files. *ansitape* will automatically pick up this information from the header records on the tape and run accordingly. If *ansitape* does not fulfill requirements, the resulting files may be run through *dd*(1M).

Multivolume support

When *ansitape* reaches the end of a tape while reading, it requests the next volume with the message "Mount continuation tape and push return". However, the tape is not checked to ensure that the correct volume was mounted.

When *ansitape* reaches the end of a tape it is writing, it requests the next volume as described above. When the new volume is online, *ansitape* initializes it with an ANSI volume header containing a volume name generated from the volume name of the first tape of the set. If the original name is fewer than six characters long, it is padded to six characters with underscores (_). Then, the last two characters of the name are replaced by a two-digit sequence number. For example, **tap** becomes **tap_02**, **mylabl** becomes **myla02**, and so forth. The sequence number is the tape's position in the set.

FILES

/dev/rmt/mt*	half-inch magnetic tape interface
/dev/rmt/ms*	quarter-inch magnetic tape interface

SEE ALSO

dd(1M) in the *UNIX System V System Administrator's Reference Manual.*
umask(2) in the *UNIX System V Programmer's Reference Manual.*

CAVEATS

The **r** (write) option cannot be used with quarter-inch archive tapes, since these tape drives cannot backspace.

The *n-th* occurrence of a file cannot be requested.

File names longer than 80 characters are truncated. This is a limitation of the ANSI labeling standard. If the tape is made without HDR3 and HDR4 labels (**3** or **i** switch), the name is limited to 17 characters.

The record size of files transferred with embedded carriage control must be a multiple of the block size.

NAME

as - common assembler

SYNOPSIS

as [*option* ...] *file-name*

DESCRIPTION

The *as* command assembles the named file. The following *options* may be specified in any order:

-o *objfile*	Put the assembly output in *objfile*. By default, the output file name is formed by removing the **.s** suffix, if it has one, from the input file name and appending a **.o** suffix.
-F *arg*	*Arg* is the string **c1**, **c2**, or **c3**. This option controls the generation of instruction fixups required for the CLIPPER C100, C200, and C300 processors. The fixups are upward-compatible but not downward-compatible. For example, code generated for C100 will run on the other two processors, but code generated for C200 or C300 will not execute on C100 processors. If downward compatibility is not required and the program is compute-intensive, removing instruction fixups may improve performance.
-n	Turn off long/short address optimization. By default, addresses are optimized.
-m	Run the *m4*(1) macro processor on the assembler input.
-R	Remove (unlink) the input file after assembly is complete.
-dl	Do not produce line number information in the object file.
-V	Write the version number of the assembler being run on the standard error output.
-Y [**md**],*dir*	Find the *m4*(1) preprocessor (**m**) and/or the file of predefined macros (**d**) in directory *dir* instead of in the customary place.

FILES

$TMPDIR/*	temporary files

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

SEE ALSO

cc(1), ld(1), a.out(4).
m4(1), nm(1), strip(1), tmpnam(3S) in the *UNIX System V Programmer's Reference Manual.*

NOTES

When possible, the assembler should be accessed through a compilation system interface program (such as *cc*(1)).

WARNINGS

If the **-m** (*m4*(1) macro processor invocation) option is used, keywords for *m4*(1) cannot be used as symbols (variables, functions, or labels) in the input file since *m4*(1) cannot distinguish assembler symbols from real *m4*(1) macros.

BUGS

The **.align** assembler directive may not work in the **.text** section when optimization is performed.

CAVEATS

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

NAME

backup - incremental file system backup

SYNOPSIS

/etc/backup [*key* [*argument* ...] *file-system*]

DESCRIPTION

backup copies to magnetic tape all files changed after a certain date in the *file-system*. The *key* specifies the date and other options about the backup. *Key* consists of characters from the set **0123456789fusdWnbc**.

0-9 This is the backup level. All files modified since the last date stored in the file **/etc/dumpdates** for the same *file-system* at lesser levels will be backed up. If no date is determined by the level, the beginning of time is assumed; thus, the option 0 causes the entire *file-system* to be backed up.

f Place the backup on the next *argument* file instead of on the tape. If the name of the file is -, *backup* writes to standard output. *rtc*(1) can be used with *backup* to backup to a remote tape device.

u If the backup completes successfully, write the date of the beginning of the backup on the file **/etc/dumpdates**. This file records a separate date for each *file-system* and each backup level. The format of **/etc/dumpdates** is readable text, consisting of one free format record per line: *file-system* name, increment level and backup date. **/etc/dumpdates** may be edited to change any of the fields if necessary.

s The size of the backup tape is specified in feet. The number of feet comes from the next *argument*. When the specified size is reached, *backup* will wait for reels to be changed. The default tape size is 2300 feet. A gap length of 0.8 inches is assumed for each write to the tape.

d The density of the tape, expressed in BPI, is taken from the next *a[illegible]-ment*. This is used in calculating the amount of tape used per reel. [illegible]e default is 1600.

W *backup* tells the operator what file systems need to be backed up. This information is gathered from the files **/etc/dumpdates** and **/etc/fstab**. For each file system in **/etc/dumpdates**, *backup* prints the most recent backup date and level, and highlights the file systems that should be backed up. If the **W** option is set, all other options are ignored and *backup* exits immediately.

w Resembles **W**, but prints only the file systems that need to be backed up.

n When *backup* requires operator attention, notify all of the operators in the group "operator".

b The number of 1K byte blocks written to the tape at a time comes from the next *argument*. This will affect how much tape is used for gaps between writes. This number cannot exceed 10 when using **/dev/rmt/rtc*** as the tape device (see *rtc*(1) and the **f** *key* above).

c Specify that a cartridge tape is being used. The default density is 8700 BPI. The default length is 600 feet. A gap length of 0 is assumed.

If no *arguments* are given, the *key* is assumed to be **9u** and a default file system is backed up to the default tape.

backup requires operator intervention on these conditions: end of tape, end of backup, tape write error, tape open error, or disk read error (if more than a threshold of 32 occur). In addition to alerting all operators implied by the **n** *key*, *backup* interacts with the operator on *backup*'s control terminal when *backup* can no longer proceed or when something is grossly wrong. All questions *backup* poses **must** be answered by typing "yes" or "no" appropriately.

Since a full backup requires much time and effort, *backup* checkpoints itself at the start of each tape volume. If writing the volume fails, *backup* will, with operator permission, restart from the checkpoint after the old tape has been rewound and removed, and a new tape has been mounted.

backup informs the operator periodically of usually low estimates of the number of blocks to write, the number of tapes the write will take, the time until completion, and the time until the tape change. The output is verbose so that others know that the terminal controlling *backup* is busy, and will be for some time.

The recommended method of performing backups is to first start with a full level 0 backup:

backup 0un

Next, active file systems are backed up daily, using a modified Tower of Hanoi algorithm with this sequence of backup levels:

3 2 5 4 7 6 9 8 9 9 ...

For the daily backups, a set of 10 tapes per backed up file system is used on a cyclical basis. Each week, a level 1 backup is performed and the daily Hanoi sequence repeats with 3. For weekly backups, a set of five tapes per backed-up file system is used. The set is also used on a cyclical basis. Each month, a level 0 backup is performed on a set of fresh tapes that is saved permanently.

FILES

/dev/dsk/s0u0p7.3	default file system to backup
/dev/rmt/0m	default tape unit to backup to
/etc/dumpdates	backup date record
/etc/fstab	backup table: file systems and frequency
/etc/group	to find group operator

SEE ALSO

restore(1), backup(4), fstab(4).

DIAGNOSTICS

backup exits with zero status on success. Startup errors are indicated with an exit code of 1; abnormal termination is indicated with an exit code of 3.

BUGS

Fewer than 32 read errors on the file system are ignored. Each reel requires a new process, so parent processes for reels already written wait until the entire tape is written.

backup with the **W** or **w** *key* does not report file systems that are not recorded in **/etc/dumpdates** even if they are listed in **/etc/fstab**.

NAME

cc - C compiler

SYNOPSIS

cc [*option* ...] *file* ...

DESCRIPTION

The *cc* command controls compilation and link editing of C and assembler source programs. The compilation process is divided into many phases. Each phase is invoked with appropriate arguments and options.

cc uses the high performance CLIPPER C compiler developed by Green Hills Software under Intergraph® contract. The CLIPPER C compiler performs optimizations not found in many other C compilers (such as the portable C compiler).

Compilation Phases

The compilation phases and their names are largely historic. Each phase is approximately implemented as a single command. There are a number of options that control the invocation of each phase. Such options use key letters to indicate a particular phase.

The phases and their key letters are:

p The preprocessor phase. This phase processes the preprocessor directives in a C source file. Preprocessor directives are given on lines whose first character is the **#** symbol. The preprocessor implements file inclusion, conditional code inclusion, macro definition, and macro expansion (see *cpp*(1)).

0 (zero) The source analysis phase. This phase analyzes the (preprocessed) source file according to the rules of the C language proper. Syntax and semantic errors are detected here. Typically, an internal or intermediate representation of the source file is built.

1 The code generation phase. This phase generates assembler code from the internal or intermediate representation.

2 The code improver phase. This optional phase examines the assembler code generated and attempts a number of improvements.

a The assembler phase. The assembler phase translates the assembler code into an object (or binary) file. See *as*(1), the "Assembler" section of the "Technical Programming Tutorial" in the *CLIX System Guide*, and the *CLIPPER User's Manual.*

l The link edit phase. Startoff routines, generated objects, and standard libraries are linked together into an image file (see *ld*(1)).

The CLIPPER C compiler implements the preprocessor, source analysis, and code generation phases in one program (**/lib/comp**). For the options that take a phase key letter, **0** indicates this program, and the key letters **p**, **1**, and **2** are ignored.

The assembler (**/bin/as**) and link editor (**/bin/ld**) implement the assembler and link editor phases, respectively.

Each input file is processed by each phase in sequence. If an error occurs in a phase, further processing of the input file that contained the error is abandoned. (The assembler will not be invoked if a compiler error occurred). Any remaining input files are compiled (or assembled), but the link edit phase is not performed.

Command Arguments

Each argument represents an option or a file name. Many options (discussed below) and three types of file names are understood. All file names and options not recognized are passed on to the link editor.

File names that end with **.c** are considered C source programs. They are compiled by applying the preprocessor through the assembler phases. Each object (relocatable binary) file is left in the current directory whose name is that of the source with **.o** substituted for **.c**. For example, compiling the file **src/prog.c** results in the file **prog.o** in the current directory.

Similarly, file names that end with **.s** are considered assembler source programs. They are processed by the assembler phase. Each object file is left in a file in the current directory whose name is that of the source with **.o** substituted for **.s**.

File names that end with **.o** are considered object files. They are passed directly to the link edit phase.

If only one **.c** or **.s** file is processed and no **.o** files are specified, the object file is normally deleted after the link edit phase completes. The object file is not deleted if the link edit phase is suppressed, an error occurs during the link edit phase, or the generated object file already existed before compilation.

The input files are processed in the left-to-right order in which they appear on the command line. The generated object files are passed to the link edit phase in the same order.

Options

Many options are intentionally undocumented. The undocumented options are disabled, obsolete, or for compiler debug only. Using undocumented options may generate poor or incorrect code. Before the description of each option and enclosed in parentheses, there may be a restriction on the use of the option. The option is only to be used when that restriction applies.

-B *string* (Obsolete; use **-Y** instead) See the description of **-t** also.

Construct path names for substitute compiler, assembler, and link editor passes by concatenating *string* with the suffixes **comp, as**, and **ld** respectively. If *string* is empty, it is assumed to be **/lib/o**.

-c Suppress the link edit phase of the compilation and force an object file to be produced even if only one program is compiled.

-C Retain comments in the preprocessor output. The default is to strip comments from the output.

-D *name* Define *name* to the preprocessor with the value 1. This is equivalent to putting the following at the top of the source file:

> **#define** *name* **1**

-D *name*=*string*
Define *name* to the preprocessor with the value *string*. This is equivalent to putting the following at the top of the source file:

> **#define** *name string*

-E See the description of **-P** also.

Do not compile the program; instead, run only the preprocessor portion of the CLIPPER C compiler and place the output on the standard output. This is useful for debugging preprocessor macros. When preprocessing for a purpose other than debugging macros, use *cpp*(1) for best performance.

-f (Ignored) Link the object program with the floating-point interpreter for systems without hardware floating-point.

-g Cause the compiler to generate additional information needed to use source language debuggers like *sdb*(1) and force the compiler to generate frame pointers for stack traces.

-ga Generate a frame pointer for stack traces. **-g** in *cc* also produces a frame pointer, but **-ga** does not produce the extra debugging information.

-I *string* File names in **#include** preprocessor directives that are not absolute (do not start with /) are searched for in the directory *string* before a default list of directories. Multiple **-I** options can be specified. They will be searched in the left to right order encountered.

-o *file-name* Place the executable binary output from the link edit phase in the file named *file-name*. If this option is not specified, the executable file will be named **a.out**. This option is ignored if **-c** or **-S** is present.

-O The **-O** option activates Green Hills optimizers that are safe to use on all programs, except for the loop optimizer.

-OM This option is equivalent to **-O** except that it also allows the optimizer to assume that memory locations do not change except by explicit stores. That is, the optimizer is guaranteed that no memory locations are I/O device registers that can be changed by external hardware and no memory locations are shared with other processes that can change them asynchronously with respect to the current process. This compile time option must be used with extreme caution (or not at all) in device drivers, operating systems, shared memory environments, and when interrupts (or CLIX signals) are present.

-OL Optimize the program to be as fast as possible even if the program must be bigger. In particular, most of the available resources are allocated to optimizations of the innermost loops. The **-OL** compile time option will perform optimizations that may make the program faster but larger. It is counterproductive to specify **-OL** on code that contains no loops or that is rarely executed as it will make the whole program larger but no faster. After experimenting with a program, it is possible to discover which modules benefit from **-OL** and which ones do not.

-OLM This option is equivalent to **-OL** and **-OM**.

-OML This option is equivalent to **-OLM**.

-p Arrange for the compiler to produce code that counts the number of times each routine is called; also, if link editing occurs, replace the standard startoff routine by one that automatically calls *monitor*(3C) at the start and arranges to write out a **mon.out** file at normal termination of execution of the object program. An execution profile can then be generated by using *prof*(1).

-P See the description of **-E** also.

Do not compile the program; instead, run just only preprocessor portion of the CLIPPER C compiler and place the output in a corresponding file suffixed with **.i**. Line control information for the next pass of the compiler is not provided. This is useful for debugging preprocessor macros. When preprocessing for a purpose other than debugging macros, use *cpp*(1) for best performance.

-S Compile the named C programs and leave the assembly language output on corresponding files suffixed with **.s**. The assembler and link edit phases are suppressed.

-t [p012al] Find only the designated phase(s) in the file whose name is constructed by a **-B** option. If an explicit **-B** option is missing, **-B /lib/n** is implied. The option **-t ""** is equivalent to **-tp012**.

-U *name* Undefine the predefined preprocessor symbol *name*. This is equivalent to putting the following at the top of the source file:

#undef *name*

-w Suppress warning diagnostics.

-W_c_**,**_arg1_**[,**_arg2_ ...**]**
Pass the listed argument(s) *argi* to phase *c* where *c* is one of **[p012al]**.

-Xa Enable ANSI-compliant compilation. With this option enabled, source code is compiled against the definition of the C language presented in the draft ANSI standard. This option causes the compiler to enforce ANSI syntax and use ANSI semantics in cases where K&R C and ANSI C conflict. The directory **/usr/include/ansi** is automatically searched for include files and, when linking, the library **/usr/lib/libansi.a** is automatically added to the default library list.

-X *n* Turn on compile time option number *n*. The available compile time options are listed below.

- **6** Allocate each enumerated type as the smallest size predefined type that allows all listed values (**char**, **short**, **int**, **unsigned char**, **unsigned short**, or **unsigned**) to be represented. The default is to allocate as an **int**.
- **9** Disable the local (peephole) optimizer.
- **18** Do not allocate programmer-defined local variables to a register unless the variables are declared **register**. This option suppresses optimizations that frustrate debuggers and *setjmp*(3C).
- **32** Display the names of files as they are opened. This is useful for finding out why the compiler cannot find an include file.
- **37** Emit a warning when dead code is eliminated.
- **39** Do not move frequently used procedure and data addresses to registers.
- **50** Push arguments on the stack. The default is to pass the first two arguments in registers. This option is not recommended because it produces a calling sequence incompatible with the rest of the CLIX System.
- **54** Inform the optimizer that no memory locations can change value asynchronously with respect to the running program. **-O2** sets this compile time option. (See **-O2** above).
- **55** Make fields of type **int**, **short**, and **char** be signed. The default is for all fields to be unsigned.
- **58** Do not put an underscore in front of the names of global variables and procedures. This option is not recommended because it produces symbols that are incompatible with the rest of the CLIX System.
- **62** (Default) The target processor is a CLIPPER microprocessor.

74	(Default) The target system is CLIX System V.
80	Disable the code hoisting optimization. This can speed compilation in some cases.
81	Allow external variables to be initialized (by turning off **extern**). Ordinarily, initialized **externs** are an error.
83	(Default) Enable the *va_type*, *va_stkarg*, *va_intregn*, *va_dblregn*, *va_argnum*, *va_regtyp*, and *va_align* intrinsic functions to support *varargs*(5). See the description of *varargs* support in the "C Language" chapter of the *CLIPPER C Reference Manual.*
84	Generate error messages for C anachronisms. By default, the old assignment operators (such as =+ and =−), initialization (**int i 1**), and references to members of other structures compile correctly but generate warning messages.
85	Generate **.bss** assembler directives for uninitialized statics. The default is to allocate initialized data.
87	Disable the optimization that deletes all code that stores into or modifies variables that are never read from.
89	Pack structures with no space between members, even if elements become inaccessible due to machine data alignment constraints.
105	Allow **#define** symbols to be redefined to the preprocessor.
164	(Unsupported) Do not stop if a code generator abort occurs or an "Internal Compiler Error" error message appears. This is occasionally useful in determining the cause of a compiler failure.
167	(Unsupported) Evaluate expressions involving only **float** operands as **float** (not **double**). Do not expand **float** arguments to **double**. Do not expand **float** return values to **double**. This option is not recommended because it produces code incompatible with the rest of the CLIX System.
168	Do not move invariant floating-point expressions out of loops.
176	Always convert computations involving floating-point values to **double**. By default, the compiler tries to shorten computations to **float** if the result would be the same.
190	Assume halfword objects are not aligned.

191	Assume word objects are not aligned.
192	Assume single-precision objects are not aligned.
193	Assume double-precision objects are not aligned.
194	Assume word objects are aligned only to halfword boundaries.
195	Assume single precision objects are aligned only to half-word boundaries.
196	Assume double precision objects are aligned only to halfword boundaries.
197	Assume double precision objects are aligned only to word boundaries.

-Y [p012alSILU], *dirname*

Use *dirname* to locate the phase(s) or directory(ies) specified by the key letter(s). The key letters **[p012al]** represent the phases described above. The additional key letters have the following meanings:

S	The directory containing the startup routines.
I	The default directory searched for the **#include** preprocessor directives.
L	The first default library directory searched (see *ld*(1)).
U	The second default library directory searched (see *ld*(1)).

If the location of a phase is being specified, the new path name for the phase will be *dirname/phasename*. The exact name used for *phasename* depends on the compiler driver used and the phase involved. See **FILES** below. If more than one **-Y** option is applied to a phase or directory, the last specification is used.

-Z *n* Turn off option number *n*. This is the reverse of the **-X** option. This option is useful if a version of the compiler has an option turned on by default and the user wants to turn it off.

-# (Subject to change) Print out the program name and command line arguments as each phase is invoked.

-## (Subject to change) Verbose like **-#**, only more so.

-### (Subject to change) Print out the program name and command line arguments for each phase, but do not actually invoke the phase.

FILES

file.c	C source input file
file.s	assembler source input file
file.o	object file; generated or input

a.out	linked output
/tmp/ctm*	temporary
/usr/tmp/ctm*	temporary
/lib/cpp	C preprocessor *cpp*(1)
/lib/comp	CLIPPER C compiler, *cc*
/bin/as	assembler, *as*(1)
/bin/ld	link editor, *ld*(1)
/lib/crt[1n].o	runtime startoff
/lib/mcrt[1n].o	profiling startoff
/lib/libc.a	standard C library; see sections (3C) and (3S) in the *UNIX System V Programmer's Reference Manual*
/usr/lib/libansi.a	library of ANSI C support functions
/usr/include/ansi/*.h	Include files containing macros and data structure definitions specific to ANSI C.
/lib/libp/lib*.a	profiled versions of libraries

SEE ALSO

adb(1), as(1), ld(1), sdb(1), exit(2).
cpp(1), prof(1), monitor(3C) in the *UNIX System V Programmer's Reference Manual.*
The C Programming Language by B. W. Kernighan.
Programming in C - A Tutorial by B. W. Kernighan.
C Reference Manual by D. M. Ritchie.
The "Release Notes" appendix of the *CLIPPER C Reference Manual.*
The Green Hills Software Users Manual C-CLIPPER.

DIAGNOSTICS

The diagnostics produced by C are self-explanatory. Occasional messages may be produced by the assembler or the link editor.

NOTES

By default, the return value from a C program is completely random. The only two guaranteed ways to return a specific value are to explicitly call *exit*(2) or to leave the function **main()** with a **return** *expression*; construct.

CAVEATS

If empty strings are given with the **-B** or **-t** options, they must be specified as separate command line arguments (e.g., **-t ""**, not **-t""**).

NAME

chmod - change mode

SYNOPSIS

chmod *mode file* ...
chmod *mode directory* ...

DESCRIPTION

The permissions of the named *files* or *directories* are changed according to *mode*, which may be symbolic or absolute. Absolute changes to permissions are stated using octal numbers as follows:

chmod *nnnn file* ...

N is a number from 0 to 7. Symbolic changes are stated using mnemonic characters as follows:

chmod *xyz*,... *file* ...

X is one or more characters corresponding to *user*, *group*, or *other*; *y* is +, −, or =, signifying permission assignment; and *z* is one or more characters corresponding to permission type.

If a named file is a symbolic link, the permissions of the referenced file (or directory) are changed, and the permissions of the symbolic link are undisturbed.

An absolute mode is given as an octal number constructed from the OR of the following modes:

4000	set user ID on execution
20#0	set group ID on execution if # is **7**, **5**, **3**, or **1**
	enable mandatory locking if # is **6**, **4**, **2**, or **0**
1000	sticky bit is turned on ((see *chmod*(2))
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0040	read by group
0020	write by group
0010	execute (search) by group
0004	read by others
0002	write by others
0001	execute (search) by others

Symbolic changes are stated using letters that correspond both to access classes and to the individual permissions. Permissions to a file may vary depending on the user identification number (UID) or group identification number (GID). Permissions are described in three sequences, each having three characters:

User	Group	Other
rwx	rwx	rwx

This example (meaning that user, group, and others all have read, write, and execute permissions for a given file) demonstrates two categories for granting permissions: the access class and the permissions themselves.

Thus, to change the mode of a file's (or directory's) permissions using *chmod* symbolic method, use the following syntax for mode:

[*who*] *operator* [*permission(s)*], ...

A command line using the symbolic method would appear as follows:

```
chmod g+rw file
```

This command would make allow group to read and write *file*.

Who can be stated as one or more of the following letters:

u	User's permissions.
g	Group's permissions.
o	Other's permissions.
a	Equivalent to **ugo** (all) and is the default if *who* is omitted.

Operator can be + to add *permission* to the *file's* mode, — to take away *permission*, or = to assign *permission* absolutely. (Unlike other symbolic operations, = has an absolute effect in that it resets all other bits.) Omitting *permission* is only useful with = to remove all permissions.

Permission is any compatible combination of the following letters:

r	Read permission.
w	Write permission.
x	Execute permission.
s	User or group set-ID is turned on.
t	Sticky bit is turned on.
l	Mandatory locking will occur during access.

Multiple symbolic modes separated by commas may be given, though no spaces may intervene between these modes. Operations are performed in the order given. Multiple symbolic letters following a single operator cause the corresponding operations to be performed simultaneously. The letter **s** is only meaningful with **u** or **g**, and **t** only works with **u**.

Mandatory file and record locking (l) refers to a file's ability to have its read or write permissions locked while a program is accessing that file. It is not possible to permit group execution and enable a file to be locked on execution at the same time. In addition, it is not possible to turn on the set-group-ID and enable a file to be locked on execution at the same time. The following examples,

```
chmod g+x,+l file
chmod g+s,+l file
```

are, therefore, illegal uses and will elicit error messages.

Only the owner of a file or directory (or the super-user) may change a file's mode. Only the super-user may set the sticky bit. To turn on a file's set-group-ID, the user's own group ID must correspond to the file's and group execution must be set.

EXAMPLES

To deny execution permission to all, the following commands are used. The absolute (octal) example permits only reading permissions.

```
chmod a-x file
chmod 444 file
```

To enable reading and writing of a file by the group and others, use one of the following:

```
chmod go=rw file
chmod 066 file
```

This causes a file to be locked during access:

```
chmod +l file
```

The last two examples enable all to read, write, and execute the file; and they turn on the set-group-ID.

```
chmod =rwx,g+s file
chmod 2777 file
```

SEE ALSO

ls(1).
chmod(2) in the *UNIX System V Programmer's Reference Manual.*

NOTES

In a Remote File Sharing environment, a user may not have the permissions that the output of the **ls** -l command implies. For more information, see the "Mapping Remote Users" section of Chapter 10 of the *UNIX System V System Administrator's Guide.*

NAME

chown, chgrp - change owner or group

SYNOPSIS

chown *owner file* ...
chown *owner directory* ...
chgrp *group file* ...
chgrp *group directory* ...

DESCRIPTION

chown changes the owner of the *files* or *directories* to *owner*. The *owner* may be either a decimal user ID or a login name found in the password file.

chgrp changes the group ID of the *files* or *directories* to *group*. The *group* may be either a decimal group ID or a group name found in the group file.

Unless either command is invoked by the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000, respectively, will be cleared.

Only the owner of a file (or the super-user) may change the owner or group of that file.

If the named file is a symbolic link, ownerships of the link itself are modified and the ownerships of the referenced file are undisturbed.

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1), group(4), passwd(4).
chown(2) in the *UNIX System V Programmer's Reference Manual.*

NOTES

In a Remote File Sharing environment, a user may not have the permissions that the output of the **ls** -l command implies. For more information see the "Mapping Remote Users" section in Chapter 10 of the *UNIX System Administrator's Guide.*

NAME

ci - check in RCS revisions

SYNOPSIS

ci [*option* ...] *file* ...

DESCRIPTION

ci stores new revisions in Revision Control System (RCS) files. Each file name ending in ",v" is interpreted to be an RCS file; all others are assumed to be working files containing new revisions. *ci* deposits the contents of each working file in the corresponding RCS file. If only a working file is given, *ci* tries to find the corresponding RCS file in the ./RCS directory and then in the current directory. For more details, see the **File Naming** section below.

For *ci* to work, the caller's login must be on the access list, except if the access list is empty or the caller is the super-user or owner of the file. To append a new revision to an existing branch, the tip revision on that branch must be locked by the caller. Otherwise, only a new branch can be created. This restriction is not enforced for the owner of the file unless locking is set to strict (see *rcs*(1)). A lock held by another user may be broken with the *rcs*(1) command.

Normally, *ci* checks whether the revision to be deposited is different from the preceding one. If it is not, *ci* either aborts the deposit (if **-q** is given) or asks whether to abort (if **-q** is omitted). A deposit can be forced with the **-f** option.

For each revision deposited, *ci* prompts for a log message. The log message should summarize the change and must be terminated with a line containing a single "." or a <CONTROL>-D. If several files are checked in, *ci* asks whether to reuse the previous log message. If the standard input is not a terminal, *ci* suppresses the prompt and uses the same log message for all files. See also **-m**.

The number of the deposited revision can be given by any of the *options* **-r**, **-f**, **-k**, **-l**, **-u**, or **-q**.

If the RCS file does not exist, *ci* creates it and deposits the contents of the working file as the initial revision (with a default number of 1.1). The access list is initialized to empty. Instead of requesting the log message, *ci* requests descriptive text (see **-t** below).

-r[*rev*] Assigns the revision number *rev* to the checked-in revision, releases the corresponding lock, and deletes the working file. This is the default. *Rev* may be symbolic, numeric, or mixed.

If *rev* is a revision number, it must be higher than the latest one on the branch to which *rev* belongs or it must start a new branch.

If *rev* is a branch rather than a revision number, the new revision is appended to that branch. The level number is obtained by incrementing the branch's tip revision number. If *rev* indicates a nonexisting branch, the branch is created with the initial revision numbered *rev*.1.

If *rev* is omitted, *ci* tries to derive the new revision number from the caller's last lock. If the caller has locked the tip revision of a branch, the new revision is appended to that branch. The new revision number is obtained by incrementing the tip revision number. If the caller locked a nontip revision, a new branch is started at that revision by incrementing the highest branch number at that revision. The default initial branch and level numbers are both 1.

If *rev* is omitted and the caller has no lock but is the owner of the file and locking is not set to *strict*, the revision is appended to the default branch (normally the trunk; see *rcs*(1) **-b**).

However, on the trunk, revisions can be appended, but not inserted.

-f[*rev*] Forces a deposit. The new revision is deposited even it does not differ from the preceding one.

-k[*rev*] Searches the working file for keyword values to determine its revision number, creation date, state, and author (see *co*(1)), and assigns these values to the deposited revision, rather than computing them locally. It also generates a default login message noting the login of the caller and the actual checkin date. This option is useful for software distribution. A revision sent to several sites should be checked in at these sites with the **-k** option to preserve the original number, date, author, and state. The extracted keyword values and the default log message may be overridden with **-r**, **-d**, **-s**, **-w**, and **-m**.

-l[*rev*] Works like **-r**, except it performs an additional *co*(1) **-l** for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is useful for saving a revision although one wants to continue editing it after the checkin.

-u[*rev*] Works like **-l**, except that the deposited revision is not locked. This is useful to process (compile) the revision immediately after checkin.

-q[*rev*] Quiet mode; diagnostic output is not printed. A revision that is not different from the preceding one is not deposited unless **-f** is given.

-d*date* Uses *date* for the checkin date and time. *Date* may be specified in free format as explained in *co*(1). Useful for lying about the checkin date and for **-k** if no date is available.

-m*msg* Uses the string *msg* as the log message for all revisions checked in.

-n*name*	Assigns the symbolic name *name* to the number of the checked-in revision. *ci* prints an error message if *name* is assigned to another number.
-N*name*	Same as **-n**, except that it overrides a previous assignment of *name*.
-s*state*	Sets the state of the checked-in revision to the identifier *state*. The default is **Exp**.
-t[*txtfile*]	Writes descriptive text into the RCS file. (Deletes the existing text). If *txtfile* is omitted, *ci* prompts the user for text supplied from the standard input, terminated with a line containing a single "." or <CONTROL>-D. Otherwise, the descriptive text is copied from the file *txtfile*. During initialization, descriptive text is requested even if **-t** is not given. The prompt is suppressed if standard input is not a terminal.
-**w***login*	Uses *login* for the author field of the deposited revision. Useful for lying about the author and for **-k** if no author is available.

File Naming

Pairs of RCS files and working files may be specified in three ways (see also the example section of *co*(1)).

1) Both the RCS file and the working file are given. The RCS file name has the form *path1/workfile*,**v** and the working file name has the form *path2/workfile*, where *path1/* and *path2/* are (possibly different or empty) paths and *workfile* is a file name.

2) Only the RCS file is given. Then, the working file is assumed to be in the current directory and its name is derived from the name of the RCS file by removing *path1/* and the suffix ",v".

3) Only the working file is given. Then *ci* looks for an RCS file with the form *path2/***RCS/***workfile*,**v** or *path2/workfile*,**v** (in this order).

If the RCS file is specified without a path in 1) and 2), *ci* looks for the RCS file first in the directory **./RCS** and then in the current directory.

File Modes

An RCS file created by *ci* inherits the read and execute permissions from the working file. If the RCS file exists, *ci* preserves its read and execute permissions. *ci* always turns off all write permissions of RCS files.

SEE ALSO

co(1), ident(1), rcs(1), rcsclean(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(4), sccstorcs(1).

Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

DIAGNOSTICS

For each revision, *ci* prints the RCS file, the working file, and the number of both the deposited and the preceding revision. The exit status always refers to the last file checked in and is 0 if the operation was successful or 1 otherwise.

NOTES

The caller of the command must have read/write permission for the directories containing the RCS file and the working file, and read permission for the RCS file itself. A number of temporary files are created. A semaphore file is created in the directory containing the RCS file. *ci* always creates a new RCS file and unlinks the old one. This strategy makes links to RCS files useless.

IDENTIFICATION

Author: Walter F. Tichy,
Purdue University, West Lafayette, IN 47907.
Copyright © 1982 by Walter F. Tichy.

NAME

clh - Intergraph network clearinghouse management program

SYNOPSIS

clh [[**-adur**] *object*]
clh [**-v** | **-bcp** [*arg*]] **-l** *object*

DESCRIPTION

clh provides a user interface for modifying and examining the Intergraph clearinghouse database. If command-line arguments are not given, *clh* provides a menu-driven interface.

The following options are available. **owned, local,** and **heard** refer to subdirectories under **/usr/lib/nodes.**

-a	Add *object* to the **owned** directory as an alias for the node name of the local machine.
-d	Delete *object* from the **owned** directory.
-u	Update *object* in the **owned** directory to the current Local Area Network (LAN) and to all LANs to which it is scoped (see *clh*(4)).
-r	Copy all entries in the **heard** directory on the machine *object* to the **heard** directory on the local machine.
-l	Look up *object* in the local clearinghouse. The clearinghouse directories **local, heard,** and **owned** will be searched in respective order, and the first occurrence of *object* will be printed.
-v	Print the entire contents of *object*.
-b [*arg*]	Look up all node names associated with the specified network address of *arg*.
-p [*arg*]	Print the property *arg* in *object*.
-c [*arg*]	Look up the address of *object* on the machine *arg*.

FILES

/usr/lib/nodes/owned	well-known node name and aliases
/usr/lib/nodes/local	local files used by the clearinghouse
/usr/lib/nodes/heard	all heard objects from the network

SEE ALSO

clh(4).
"XNS Network Programming Tutorial" in the *CLIX System Guide*.

NAME

co - check out RCS revisions

SYNOPSIS

co [*option* ...] *file* ...

DESCRIPTION

co retrieves a revision from each Revision Control System (RCS) file and stores it into the corresponding working file. Each file name ending in ",v" is assumed to be an RCS file; all other files are assumed to be working files. If only a working file is given, *co* tries to find the corresponding RCS file in the directory **./RCS** and then in the current directory. For more details, see the **File Naming** section below.

Revisions of an RCS file may be checked out locked or unlocked. Locking a revision prevents overlapping updates. A revision checked out for reading or processing (compiling) need not be locked. A revision checked out for editing and later checkin must normally be locked. *co* with locking fails if the revision to be checked out is currently locked by another user. (A lock may be broken with the *rcs*(1) command.) *co* with locking also requires the caller to be on the access list of the RCS file unless the caller is the file owner or the super-user, or the access list is empty. *co* without locking is not subject to access list restrictions, and is not affected by locks.

A revision is selected by *options* for revision or branch number, checkin date/time, author, or state. When the selection *options* are applied in combination, *co* retrieves the latest revision that satisfies all of them. If no selection *option* is specified, *co* retrieves the latest revision on the default branch. (Normally the trunk, see *rcs*(1) **-b**.) A revision or branch number may be attached to any of the *options* **-f**, **-l**, **-p**, **-q**, **-r**, or **-u**. The *options* **-d** (date), **-s** (state), and **-w** (author) retrieve from a single branch, the *selected* branch, that is either specified by **-f**, **-l**, **-p**, **-q**, **-r**, **-u**, or the default branch.

A *co* command applied to an RCS file with no revisions creates a zero-length working file. *co* always performs keyword substitution (see below).

-r[*rev*] Retrieves the latest revision whose number is less than or equal to *rev*. If *rev* indicates a branch rather than a revision, the latest revision on that branch is retrieved. If *rev* is omitted, the latest revision on the default branch (see *rcs*(1) **-b**) is retrieved. *Rev* is composed of one or more numeric or symbolic fields separated by ".". The numeric equivalent of a symbolic field is specified with the **-n** option of the commands *ci*(1) and *rcs*(1).

-l[*rev*] Same as **-r** except that it also locks the retrieved revision for the caller. See **-r** to see how revision number *rev* is handled.

-u[*rev*] Same as **-r** except that it unlocks the retrieved revision (if it was locked by the caller). If *rev* is omitted, **-u** retrieves the latest revision locked by the caller. If no such lock exists, it retrieves the latest revision on the default branch.

-f[*rev*] Forces the working file to be overwritten. Useful when used with **-q**. See also the section on **File Modes** below.

-p[*rev*] Prints the retrieved revision on the standard output rather than storing it in the working file. This option is useful when *co* is part of a pipe.

-q[*rev*] Quiet mode; diagnostics are not printed.

-d*date* Retrieves the latest revision on the selected branch whose checkin date/time is less than or equal to *date*. The date and time may be given in free format and are converted to local time. Examples of formats for *date* include the following:

```
22-April-1982, 17:20-CDT
2:25 AM, Dec. 29, 1983
Tue-PDT, 1981, 4pm Jul 21            (free format)
Fri, April 16 15:52:25 EST 1982      (output of ctime)
```

Most fields in the date and time may be defaulted. *co* determines the defaults in the order of year, month, day, hour, minute, and second (most to least significant). At least one of these fields must be provided. For omitted fields with higher significance than the highest provided field, the current values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date "20, 10:30" defaults to 10:30:00 of the 20th of the current month and year. The date/time must be quoted if it contains spaces.

-s*state* Retrieves the latest revision on the selected branch whose state is set to *state*.

-w[*login*] Retrieves the latest revision on the selected branch that was checked in by the user with login name *login*. If the argument *login* is omitted, the caller's login is assumed.

-j*joinlist* Generates a new revision that is the join of the revisions on *joinlist*. *Joinlist* is a comma-separated list of pairs of the form *rev2:rev3*, where *rev2* and *rev3* are (symbolic or numeric) revision numbers. For the initial such pair, *rev1* denotes the revision selected by the above options **-r**, ... , **-w**. For all other pairs, *rev1* denotes the revision generated by the previous pair. (Thus, the output of one join becomes the input to the next.)

For each pair, *co* joins revisions *rev1* and *rev3* with respect to *rev2*. This means that all changes that transform *rev2* into *rev1* are applied to a copy of *rev3*. This is useful if *rev1* and *rev3* are the ends of two branches that have *rev2* as a common ancestor. If *rev1*<*rev2*<*rev3* on the same branch, joining generates a new revision that is like *rev3*, but with all changes that lead from *rev1* to *rev2* undone. If changes from *rev2* to *rev1* overlap changes from *rev2* to *rev3*, *co* prints a warning and includes the overlapping sections, delimited by the lines **<<<<<<<**

rev1, **=======**, and **>>>>>>>** *rev3*.

For the initial pair, *rev2* may be omitted. The default is the common ancestor. If any arguments indicate branches, the latest revisions on those branches are assumed. The **-l** and **-u** options lock or unlock *rev1*.

Keyword Substitution

Strings with the form **$***keyword***$** and **$***keyword*: ...**$** embedded in the text are replaced with strings with the form **$***keyword*: *value* **$**, where *keyword* and *value* are pairs listed below. Keywords may be embedded in literal strings or comments to identify a revision.

Initially, the user enters strings with the form **$***keyword***$**. On checkout, *co* replaces these strings with strings with the form **$***keyword*: *value* **$**. If a revision containing strings with the latter form is checked back in, the value fields will be replaced during the next checkout. Thus, the keyword values are automatically updated at checkout.

Keywords and their corresponding values are as follows:

$Author$	The login name of the user who checked in the revision.
$Date$	The date and time the revision was checked in.
$Header$	A standard header containing the full path name of the RCS file, the revision number, the date, the author, the state, and the locker (if locked).
$Id$	Same as **$Header$** except that the RCS file name has no path.
$Locker$	The login name of the user who locked the revision (empty if not locked).
$Log$	The log message supplied during checkin, preceded by a header containing the RCS file name, the revision number, the author, and the date. Existing log messages are NOT replaced. Instead, the new log message is inserted after **$Log:** ...**$**. This is useful for accumulating a complete change log in a source file.
$RCSfile$	The name of the RCS file without path.
$Revision$	The revision number assigned to the revision.
$Source$	The full path name of the RCS file.
$State$	The state assigned to the revision with *rcs*(1) **-s** or *ci*(1) **-s**.

File Naming

Pairs of RCS files and working files may be specified in three ways (see also the example section).

1) Both the RCS file and the working file are given. The RCS file name has the form *path1/workfile***,v** and the working file name has the form *path2/workfile*, where *path1/* and *path2/* are (possibly different or empty) paths and *workfile* is a file name.

2) Only the RCS file is given. Then, the working file is created in the current directory and its name is derived from the name of the RCS file by removing *path1/* and the suffix ",v".

3) Only the working file is given. Then, *co* looks for an RCS file with the form *path2/***RCS***/workfile***,v** or *path2/workfile***,v** (in this order).

If the RCS file is specified without a path in 1) and 2), *co* looks for the RCS file first in the directory **./RCS** and then in the current directory.

File Modes

The working file inherits the read and execute permissions from the RCS file. In addition, the owner write permission is turned on, unless the file is checked out unlocked and locking is set to *strict* (see *rcs*(1)).

If a file with the name of the working file exists and has write permission, *co* aborts the checkout if **-q** is given or asks whether to abort if not. If the existing working file is not writable or **-f** is given, the working file is deleted without asking.

EXAMPLES

Suppose the current directory contains a subdirectory **RCS** with an RCS file **io.c,v**. Then, all of the following commands retrieve the latest revision from **RCS/io.c,v** and store it in **io.c**.

```
co io.c;            co RCS/io.c,v;
co io.c,v;          co io.c RCS/io.c,v;
co io.c io.c,v;     co RCS/io.c,v io.c;
co io.c,v io.c;
```

SEE ALSO

ci(1), ident(1), rcs(1), rcsclean(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(4), sccstorcs(1).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

DIAGNOSTICS

The RCS file name, the working file name, and the revision number retrieved are written to the diagnostic output. The exit status always refers to the last file checked out and is 0 if the operation was successful or 1 otherwise.

NOTES

The caller of the command must have write permission in the working directory, read permission for the RCS file, and either read permission (for reading) or read/write permission (for locking) in the directory that contains the RCS file.

A number of temporary files are created. A semaphore file is created in the directory of the RCS file to prevent simultaneous update.

WARNINGS

The option **-d** gets confused in some circumstances, and accepts no date before 1970.

Links to the RCS and working files are not preserved.

The expansion of keywords cannot be suppressed except by writing them differently. In nroff and troff, this is done by embedding the null-character "\&" in the keyword.

BUGS

The option -j does not work for files that contain lines with a single ".".

IDENTIFICATION

Author: Walter F. Tichy,
Purdue University, West Lafayette IN, 47907.
Copyright © 1982 by Walter F. Tichy.

NAME

compress, uncompress, zcat - compress and expand data

SYNOPSIS

compress [**-f**] [**-v**] [**-c**] [**-V**] [**-d**] [**-b** *maxbits*] [*file* ...]
uncompress [**-f**] [**-v**] [**-c**] [**-V**] [*file* ...]
zcat [**-V**] [*file* ...]

DESCRIPTION

compress reduces the size of the named files using adaptive Lempel-Ziv coding. Whenever possible, each file *file* is replaced by one with the form *file*.**Z**, while keeping the same ownership modes, access and modification times. If no files are specified, the standard input is compressed to the standard output. Compressed files can be restored to their original form using *uncompress*, *zcat*, or **compress -d**.

The **-f** option forces compression of *file*. This is useful for compressing an entire directory, even if some of the files do not actually shrink. If **-f** is not given and *compress* is run in the foreground, the user is prompted as to whether an existing file should be overwritten.

The **-c** option makes *compress* and *uncompress* write to the standard output; no files are changed. The nondestructive behavior of *zcat* is identical to that of **uncompress -c**.

compress uses the modified Lempel-Ziv algorithm popularized in "A Technique for High Performance Data Compression", Terry A. Welch, *IEEE Computer*, vol. 17, no. 6 (June 1984), pp. 8-19. Common substrings in the file are first replaced by 9-bit codes 257 and up. When code 512 is reached, the algorithm switches to 10-bit codes and continues to use more bits until the limit specified by the **-b** flag is reached (default 16). *Maxbits* must be between 9 and 16. The default can be changed in the source to allow *compress* to be run on a smaller machine.

After the *maxbits* limit is attained, *compress* periodically checks the compression ratio. If it is increasing, *compress* continues to use the existing code dictionary. However, if the compression ratio decreases, *compress* discards the table of substrings and rebuilds it from scratch. This allows the algorithm to adapt to the next "block" of the file.

Note that the **-b** flag is omitted for *uncompress*, since the *maxbits* parameter specified during compression is encoded within the output, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is attempted.

The amount of compression obtained depends on the size of the input, the number of bits per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50-60%. Compression is generally much better than that achieved by Huffman coding or adaptive Huffman coding, and takes less time to compute.

Under the **-v** option, a message is printed yielding the percentage of reduction for each file compressed.

If the **-V** option is specified, the current version and compile options are printed on **stderr**.

DIAGNOSTICS

Exit status is normally 0; if the last file is larger after (attempted) compression, the status is 2; if an error occurs, exit status is 1.

Usage: compress [-dfvcV] [-b *maxbits*] [*file* ...]
: Invalid options were specified on the command line.

Missing maxbits
: *Maxbits* must follow **-b**.

file: not in compressed format
: The file specified to *uncompress* has not been compressed.

file: compressed with *xx* bits, can only handle *yy* bits
: *File* was compressed by a program that could deal with a larger *maxbits* than the compress code on this machine. Recompress the file with smaller *maxbits*.

file: already has .Z suffix -- no change
: The file is assumed to be already compressed. Rename the file and try again.

file: filename too long to tack on .Z
: *File* cannot be compressed because its name is longer than 12 characters. Rename and try again.

file already exists; do you wish to overwrite (y or n)?
: Respond "y" if the output file should be replaced; "n" if not.

uncompress: corrupt input
: A SIGSEGV violation was detected which usually means that the input file has been corrupted.

Compression: *xx.xx*%
: Percentage of the input saved by compression. (Relevant only for **-v**.)

— not a regular file: unchanged
: When the input file is not a regular file, (e.g., a directory), it is left unaltered.

— has *xx* other links: unchanged
: The input file has links; it is left unchanged. See *ln*(1) for more information.

— file unchanged
: No savings are achieved by compression. The input remains virgin.

BUGS

Although compressed files are compatible between machines with large memory, **-b12** should be used for file transfer to architectures with a small process data space (64K bytes or less, as exhibited by the DEC™ PDP™ series, the Intel 80286, etc.)

NAME

cpflop - copy floppy disk

SYNOPSIS

cpflop [**-ls**] [**-n** *numcopies*]

DESCRIPTION

cpflop duplicates a floppy disk using a single floppy drive. In the default operation mode, with no command line parameters or input/output redirection, *cpflop* copies one floppy to another, prompting for insertion of the source and destination floppies. If **stdin** is a terminal or **/dev/null**, *cpflop* assumes that the source for the copy is a floppy and prompts for the source disk to be inserted. **Stdout** is investigated in the same way to determine the destination of the copy. If the file is not a terminal or **/dev/null**, the file associated with **stdin** and/or **stdout** is accessed instead of the floppy. No prompt is issued for devices other than floppy.

The options supported are as follows:

-l	Floppy is low density (720 blocks).
-n *numcopies*	*Numcopies* is the number of copies of the source floppy to make.
-s	No prompting for floppy insertion. This option is not valid if both the source and destination are floppies. The **-s** option assumes that the amount of data involved can be contained on one floppy.

FILES

/dev/rdsk/floppy

CAVEATS

cpflop does not support multiple-sequenced volumes. It transfers a maximum of 2400 blocks (720 for low density) from the source to the destination, repeating this action with the same source data if multiple copies are requested.

NAME

cpio - copy file archives in and out

SYNOPSIS

cpio -o[acBvV] [-C *bufsize***] [[-O** *file***] [-M** *message***]]**

cpio -i[BcdmrtuvVfsSb6k] [-C *bufsize***] [[-I** *file***] [-M** *message***]]** [*pattern* ...]

cpio -p[adlmuvV] *directory*

DESCRIPTION

cpio -o (copy out) reads the standard input to obtain a list of path names and copies those files on the standard output with path name and status information. Output is padded to a 512-byte boundary by default.

cpio -i (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio -o**. Only files with names that match *patterns* are selected. *Patterns* are regular expressions given in the file name generating notation of *sh*(1). In *patterns*, meta-characters **?**, *****, and **[...]** match the slash (/) character, and backslash (\) is an escape character. A **!** meta-character means NOT. (For example, the **!abc*** pattern excludes all files that begin with **abc**.) Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is ***** (select all files). Each *pattern* must be enclosed in double quotes; otherwise, the name of a file in the current directory is used. Extracted files are conditionally created and copied in the current directory tree based on the options described below. The permissions of the files will be those of the previous **cpio -o**. The file owner and group will be the current user's unless the user is super-user, which causes *cpio* to retain the file owner and group of the previous **cpio -o**.

If **cpio -i** tries to create a file that exists and the existing file is the same age or newer, *cpio* will output a warning message and not replace the file. (The **-u** option can be used to unconditionally overwrite the existing file.)

cpio -p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied in the destination *directory* tree based on the options described below.

The meanings of the available options are as follows:

-a	Reset access times of input files after they have been copied. Access times are not reset for linked files when **cpio -pla** is specified.
-b	Reverse the order of the bytes within each word. Use **-b** only with the **-i** option.
-B	Input/output is to be blocked 5,120 bytes to the record. The default buffer size is 512 bytes when this option and the **-C** option are not used. (**-B** does not apply to the pass option; **-B** is meaningful only with data directed to or from a character special device, e.g., **/dev/rmt/0m**.)

-c	Write header information in ASCII character form for portability. Always use this option when origin and destination machines are different types.
-C *bufsize*	Input/output will be blocked *bufsize* bytes to the record, where *bufsize* is replaced by a positive integer. The default buffer size is 512 bytes when this and **-B** options are not used. **-C** does not apply to the pass option; **-C** is meaningful only with data directed to or from a character special device (e.g., **/dev/rmt/0m**.)
-d	Directories will be created as needed.
-f	Copy all *files* except those in *patterns*. (See the paragraph on **cpio -i** for a description of *patterns*.)
-I *file*	Read the contents of *file* as input. If *file* is a character special device, when the first medium is full, replace the medium and type a carriage return to continue to the next medium. Use only with the **-i** option.
-k	Attempt to skip corrupted file headers and I/O errors that may be encountered. To copy files from a medium that is corrupted or out of sequence, this option lets only files with good headers be read. (For *cpio* archives that contain other *cpio* archives, if an error is encountered, *cpio* may terminate prematurely. *cpio* will find the next good header, which may be for a smaller archive, and terminate when the smaller archive's trailer is encountered.) Used only with the **-i** option.
-l	When possible, link files rather than copying them. Usable only with the **-p** option.
-m	Retain previous file modification time. This option is ineffective on directories and symbolic links that are being copied.
-M *message*	Define a message to use when switching media. When using the **-O** or **-I** options and specifying a character special device, this option can be used to define the message printed when reaching the end of the medium. One **%d** can be placed in the message to print the sequence number of the next medium needed to continue.
-O *file*	Direct the output of *cpio* to *file*. If *file* is a character special device, when the first medium is full, replace the medium and type a carriage return to continue to the next medium. Use only with the **-o** option.
-r	Interactively rename files. If the user types a null line, the file is skipped. If the user types a "." the original path name will be copied. (Not available with **cpio -p**.)

-s	Swap bytes within each half word. Use only with the **-i** option.
-S	Swap halfwords within each word. Use only with the **-i** option.
-t	Print a table of contents for the input. No files are created.
-u	Copy unconditionally. (Normally, an older file will not replace a newer file with the same name.)
-v	Verbose. Print a list of file names. When used with the **-t** option, the table of contents resembles the output of an **ls -l** command (see *ls*(1)).
-V	Special verbose. Print a dot for each file seen. This assures the user that *cpio* is working without printing all file names.
-6	Process an old (i.e., UNIX System Sixth Edition format) file. Use only with the **-i** option.

If *cpio* reaches the end of medium (such as end of a tape), when writing to (**-o**) or reading from (**-i**) a character special device and **-O** and **-I** are not used, *cpio* will print the message:

If you want to go on, type device/file name when ready.

To continue, the medium must be replaced and the character special device name (such as **/dev/rmt/0m**) and a carriage return typed. The user may want to continue by directing *cpio* to use a different device. For example, if two tape drives are available, it may be desirable to switch between them so *cpio* can proceed while tapes are being changed. (A carriage return alone causes the *cpio* process to exit.)

EXAMPLES

The following examples show three uses of *cpio*.

When standard input is directed through a pipe to **cpio -o**, it groups the files so they can be directed (>) to a single file (*../newfile*). The **-c** option ensures that the file can be ported to other machines. Instead of *ls*(1), *find*(1), *echo*(1), or *cat*(1) could be used to pipe a list of names to *cpio*. Output could be directed to a device instead of a file.

ls | cpio -oc > *../newfile*

cpio -i uses the output file of **cpio -o** (directed through a pipe with **cat** in the example), extracts the files that match the patterns (**memo/a1**, **memo/b***), creates directories below the current directory as needed (**-d** option), and places the files in the appropriate directories. The **-c** option is used when the file is created with a portable header. If no patterns were given, all files from *newfile* would be placed in the directory.

cat *newfile* **| cpio -icd "memo/a1" "memo/b*"**

cpio -p copies or links (**-l** option) the file names piped to it to another directory (**newdir** in the example). The **-d** option says to create directories as needed. The **-m** option says to retain the modification time. (It is important to use the **-depth** option of *find*(1) to generate path names for *cpio*. This eliminates problems *cpio* could have trying to create files under read-only directories.)

```
find . -depth -print | cpio -pdlmv newdir
```

SEE ALSO

find(1), ls(1), scpio(1).
sh(1), tar(1), ar(4), cpio(4) in the *UNIX System V Programmer's Reference Manual*.
cat(1), echo(1) in the *UNIX System V User's Reference Manual*.

NOTES

cpio assumes four-byte words.

Path names are restricted to 256 characters.

Only the super-user can copy special files.

Blocks are reported in 512-byte quantities.

If a file has 000 permissions, contains more than 0 characters, and the user is not root, the file will not be saved or restored.

NAME

crm - CLIX Resource Monitor

SYNOPSIS

/usr/ip32/crm/crm.sh

/usr/ip32/crm/crm.server

DESCRIPTION

crm, the CLIX Resource Monitor (CRM), invokes a menu-driven interface for monitoring the CLIX operating system. *crm* monitors either the system as a whole or individual processes and provides either alphanumeric displays based on the curses facilities or graphics displays based on Environ V facilities. On a graphics system, typing the first line of the synopsis invokes *crm* from the command line. The second line is used to invoke *crm* on a non-graphics system.

The initial *crm* window allows access to online instructions for using *crm*, entering the System Monitor and Process Monitor menus, or exiting *crm*. Use arrow keys to scroll through the choices and <RETURN> to execute. The following describes the choices available:

Instructions	Explain how to use the *crm* labels at the bottom of the *crm* window, menus, and forms.
System Monitors	Provide information about the system in areas such as I/O activity and file, memory, and CPU use. *crm* provides the following system monitors: Monitor Parameters (*monparam*(1)) Top Fault Monitor (*topfault*(1)) Top Memory Monitor (*topmem*(1)) Top CPU Monitor (*topcpu*(1)) Top I/O Monitor (*topio*(1)) Top Sys Monitor (*topsys*(1)) Show Open Files (*showfiles*(1)) Show Memory Usage (*showmemory*(1))
Process Monitors	Provide the capability to profile a process and show its paging, I/O, system call, and instruction execution. *crm* provides the following process monitors: Profiler (*watcher*(1)) Memory Monitor (*monregion*(1)) Process Monitor (*monproc*(1))
Exit	Exit *crm*.

System Monitors

Each of the system monitors can be executed from the *crm* menus or from the command line by entering the command in parentheses. However, the only way to execute these monitors (except *topsys*(1)) in graphics-based format is from the command line. *topsys*(1) displays only in graphics-based format.

The following choices are available from the System Monitor menu:

Change Defaults
The change defaults menu option allows the user to change the defaults for the remaining menu options.

The following defaults can be changed:

Sample Interval	Specify how frequently the monitor samples and displays information.
Input File	Read the data from the input file. The input file must have been previously created as a *crm* output file. A - for the input file reads input from **stdin**.
Output File	Direct output to the output file. A - for the output file directs output to **stdout**.
Graphic Windows	Invoke graphically oriented windows such as *topsys*(1).
Learn Mode	Display the command and options used to execute a monitor.
Separate Windows	Invoke a window separate from the *fmli* window and run the selected process in the separate window.

Monitor Parameters
Execute *monparam*(1).

Top Fault Monitor
Execute *topfault*(1).

Top Memory Monitor
Execute *topmem*(1).

Top CPU Monitor
Execute *topcpu*(1).

Top I/O Monitor
Execute *topio*(1).

Top System Monitor
Execute *topsys*(1).

Show Open Files
Execute *showfiles*(1).

Show Memory Usage
Execute *showmemory*(1).

Process Monitors

All *crm* process monitors display in curses-based format by default. However, the memory and process monitor can also display in a graphics-based format by selecting an option from *crm* menus. Select the Delete icon to exit from graphics-based monitors; press <CONTROL>-C to exit from curses-based Profiler and Q or X to exit from curses-based Memory and Process monitors.

crm process monitors may be executed through the *crm* menus or from the command line.

The following choices are available from the process monitor menu:

Select Process to Monitor
: Before a process can be monitored, the user must specify the process to monitor. The user can key in **ps -e** at the system prompt to determine the name or process ID (PID) of processes running on the system.

: Then, to select the process to monitored, the user chooses the Select Process to Monitor option from the main process monitor menu. A Change Default options form appears. One of the first four fields must be completed to specify which process to monitor. The rest of the fields are optional. A brief description of each field follows:

Name of program to monitor
: Allow the user to enter the process name of the process to monitor.

PID
: Allow the user to enter the PID of an active process to monitor.

program to execute
: Allow the user to enter the path name (and options) of a program to execute and monitor simultaneously.

Pre-recorded File
: Allow the user to enter the file name (path name) of a previously recorded monitoring session.

Output File
: Allow the user to enter the file name (path name) of a file where the monitoring session will be recorded.

Separate Windows
: Allow the user to execute a monitor in graphics-based format when set to Y. This field applies only to the memory and process monitors; the Profiler does not run in graphics-based format. This option should be set to N when an attempt is made to run Profiler; otherwise, it will not execute.

Sample Interval
: Allow the user to define how frequently (in seconds) a monitor will gather information and update the monitor fields. Enter a positive number in this field. This field applies only to the process monitor (*monproc*(1)).

Learn mode
Display the command and options used to execute a monitor.

Select Profiler Options
The Profiler monitors the page faults and system calls of a specified process. Before the Profiler is run, page faults and system calls must be enabled. Choose Select Profile Options from the main process monitor menu. To accept the default values for each option listed on the Page Faults and System Calls forms, press the SAVE key (<PF3>).

Select the Enable Page Faults option. See *topfault*(1) for a description of the Demand Zero, Swap, Cache, File, Copy on Write, and Steal fields. The last three fields on the form are described as follows:

Starting Virtual Address
Ending Virtual Address
Allow the user to monitor faults occurring only at certain addresses in the process. These fields allow the user to define the section of the process in which faults will be watched.

Maximum Samples
Allow the user to define the number of samples for the monitor to collect. This definition may prevent the monitor from running indefinitely.

Select the Enable System Calls option. A description of the fields follows:

All System Calls
Direct the Profiler to report all system calls. The default setting is Y.

I/O
Direct the Profiler to ignore all system calls except for I/O calls. The default setting is N.

Summary Only
Direct the monitor to print only a summary of system calls when the monitoring interval is complete instead of listing all system calls as they are encountered. The default setting is N.

Run Profiler
Execute *watcher*(1).

Run Memory Monitor
Execute *monregion*(1).

Run Process Monitor
Execute *monproc*(1).

SEE ALSO

monparam(1), monproc(1), monregion(1), showfiles(1), showmemory(1), topcpu(1), topfault(1), topio(1), topmem(1), topsys(1), watcher(1).

WARNINGS

Sending raw data to a file can create a very large file.

NAME

csh - a shell (command interpreter) with C-like syntax

SYNOPSIS

csh [**-cefinstvVxX**] [*arg* ...]

DESCRIPTION

csh is a first implementation of a command language interpreter incorporating a history mechanism (see **History Substitutions**), job control facilities (see **Jobs**), interactive file name and user name completion (see **File Name Completion**), and a C-like syntax.

csh begins by executing commands from the file **.cshrc** in the home directory of the invoker. If this is a login shell, it also executes commands from the file **.login** there.

Normally, the shell will then begin reading commands from the terminal, prompting with "% ". Argument Processing and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into "words". This sequence of words is placed on the command history list and then parsed. Finally, each command in the current line is executed.

When a login shell terminates, it executes commands from the file **.logout** in the user's home directory.

Lexical Structure

The shell splits input lines into words at blanks and tabs with the following exceptions: the characters **&**, **|**, **;**, **<**, **>**, **(**, and **)** form separate words. If doubled in **&&**, **||**, **<<**, or **>>** these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning by preceding them with \. A newline preceded by a \ is equivalent to a blank.

In addition, strings enclosed in matched pairs quotations, **\` \`**, **' '**, or **" "**, form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within **' '** or **" "**, a newline preceded by a \ gives a true newline character.

When the shell's input is not a terminal, the character **#** introduces a comment that continues until the end of the input line. It does not have this special meaning when preceded by \ and in quotations using **\` \`**, **' '**, and **" "**.

Commands

A simple command is a sequence of words, with the first specifying the command to be executed. A simple command or a sequence of simple commands separated by **|** characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by **;**, and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an **&**.

Any of the above may be placed in **( )** to form a simple command (which may be a component of a pipeline, etc.). Pipelines can also be separated with **||** or **&&** indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See **Expressions.**)

Jobs

The shell associates a job with each pipeline. It keeps a table of current jobs (printed by the **jobs** command) and assigns them small integer numbers. When a job is started asynchronously with **&**, the shell prints a line that looks like the following, indicating that the job started asynchronously was job number 1 and had one (top-level) process with process ID 1234.

```
[1] 1234
```

If a job is running and the user wishes to do something else, <CONTROL>-Z may be pressed, which sends a STOP signal to the current job. The shell will then normally indicate that the job has been "Stopped", and print another prompt. the state of this job can then be manipulated, putting it in the background with the **bg** command, or run other commands and then eventually bring the job back to the foreground with the foreground command **fg**. A <CONTROL>-Z takes effect immediately and, like an interrupt, pending output and unread input are discarded when it is typed.

A job running in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command **stty tostop**. If this tty option is set, background jobs will stop when they try to produce output as they do when they try to read input.

Jobs in the shell can be referred to in several ways. The character **%** introduces a job name. To refer to job number 1, it can be named **%1**. Naming a job brings it to the foreground; thus, **%1** is a synonym for **fg %1**, bringing job 1 to the foreground. Similarly, saying **%1 &** resumes job 1 in the background. Jobs can also be named by prefixes of the string typed to start them if these prefixes are unambiguous; thus, **%ex** would normally restart a suspended *ex*(1) job, if only one suspended job's name began with the string "ex". Saying **%?***string*, which specifies a job whose text contains *string* if only one such job exists, is also possible.

The shell maintains a status of the current and previous jobs. In output pertaining to jobs, the current job is marked with a + and the previous job with a —. The abbreviation **%+** refers to the current job and **%—** refers to the previous job. For close analogy with the syntax of the history mechanism (described below), **%%** is also a synonym for the current job.

Status Reporting

This shell learns immediately when a process changes state. It normally informs the user when a job becomes blocked so that no further progress is possible. However, this information is only received just before it prints a prompt. This is done so that it does not otherwise disturb the user's work. If, however, the shell variable *notify* is set, the shell will notify the user immediately of changes of background job status. Also, a shell command, **notify**, marks a single process so that its status changes will be immediately reported. By default, **notify** marks the current process; simply entering **notify** after starting a background job marks it.

When trying to leave the shell while jobs are stopped, the user will be warned that "You have stopped jobs." The **jobs** command may be used to see what they are. If the user does this or immediately tries to exit again, the shell will not give a second warning, and the suspended jobs will be terminated.

File Name Completion

When the file name completion feature is enabled by setting the shell variable *filec* (see **set**), *csh* will interactively complete file names and user names from unique prefixes when they are input from the terminal followed by the escape character (the <ESC> key, or <CONTROL>-[). For example, if the current directory contains the following:

```
DSC.OLD     bin        cmd       lib      xmpl.c
DSC.NEW     chaosnet   cmtest    mail     xmpl.o
bench       class      dev       mbox     xmpl.out
```

and the input is

```
% vi ch<ESC>
```

csh will complete the prefix "ch" to the only matching file name **chaosnet**, changing the input line to the following:

```
% vi chaosnet
```

However, given

```
% vi D<ESC>
```

csh will only expand the input to

```
% vi DSC.
```

and will sound the terminal bell to indicate that the expansion is incomplete, since two file names match the prefix "D".

If a partial file name is followed by the end-of-file character (usually <CONTROL>-D), then, instead of completing the name, *csh* will list all file names matching the prefix. For example, the input

```
% vi D<CONTROL>-D
```

causes all files beginning with "D" to be listed:

```
DSC.NEW     DSC.OLD
```

while the input line remains unchanged.

The same system of escape and end-of-file can also be used to expand partial user names if the word to be completed (or listed) begins with the character

~. For example, typing

```
cd ~ro<CONTROL>-D
```

may produce the expansion

```
cd ~root
```

The use of the terminal bell to signal errors or multiple matches can be inhibited by setting the variable *nobeep*.

Normally, all files in the directory are candidates for name completion. Files with certain suffixes can be excluded from consideration by setting the variable *fignore* to the list of suffixes to be ignored. Thus, if the command

```
% set fignore = (.o .out)
```

was entered, then typing

```
% vi x<ESC>
```

would result in the completion to

```
% vi xmpl.c
```

ignoring the files **xmpl.o** and **xmpl.out**. However, if the only completion possible requires not ignoring these suffixes, they are not ignored. In addition, *fignore* does not affect the listing of file names by <CONTROL>-D. All files are listed regardless of their suffixes.

Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur.

History Substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence. History substitutions begin with the character **!** and may begin anywhere in the input stream (with the proviso that they do not nest.) This **!** may be preceded by a \ to prevent its special meaning; for convenience, a **!** is passed unchanged when it is followed by a blank, tab, newline, = or (. (History substitutions also occur when an input line begins with ^. This special abbreviation will be described later.) Any input line that contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

Commands input from the terminal that consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands in the input stream. The size of the history list is controlled by the *history* variable; the previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

For definiteness, the following output from the **history** command should be considered:

```
     9  write michael
    10  ex write.c
    11  cat oldwrite.c
    12  diff *write.c
```

The commands are shown with their event numbers. Using even numbers is usually not necessary, but the current event number can be made part of the prompt by placing an **!** in the *prompt* string.

With the current event 13, previous events can be referred to by event number as in **!11**, relatively as in **!-2** (referring to the same event), by a prefix of a command word as in **!d** for event 12 or **!wri** for event 9, or by a string contained in a word in the command as in **!?mic?** (also referring to event 9). These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case, **!!** refers to the previous command; thus **!!** alone is essentially a **redo**.

To select words from an event, The event specification can be followed by a : and a *designator* for the desired words. The words of an input line are numbered from 0, the first (usually a command) word being 0, the second word (first argument) being 1, etc. The basic word *designators* are as follows:

0	first (command) word
n	*n*th argument
^	first argument, i.e., **1**
$	last argument
%	word matched by (immediately preceding) ?*s*? search
x-*y*	range of words
-*y*	abbreviates 0-*y*
*****	abbreviates **^-$**, or nothing if only 1 word in event
*x******	abbreviates *x*-**$**
x-	like *x****** but omitting word **$**

The : separating the event specification from the word designator can be omitted if the argument selector begins with a **^**, **$**, *****, -, or **%**. After the optional word designator, a sequence of modifiers can be placed, each preceded by a :. The following modifiers are defined:

h	Remove a trailing path name component, leaving the head.
r	Remove a trailing *.xxx* component, leaving the root name.
e	Remove all but the extension *.xxx* part.
s/*l***/***r***/**	Substitute *l* for *r*.
t	Remove all leading path name components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, as in **g&**.
p	Print the new command, but do not execute it.
q	Quote the substituted words, preventing further substitutions.

x Like **q**, but break into words at blanks, tabs, and newlines.

Unless preceded by a **g**, the modification is applied only to the first modifiable word. With substitutions, an error results when no word is applicable.

The left-hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of /; a \ quotes the delimiter in the *l* and *r* strings. The character & in the right-hand side is replaced by the text from the left. A \ quotes & also. A null *l* uses the previous string either from an *l* or from a contextual scan string *s* in !?*s*?. The trailing delimiter in the substitution may be omitted if a newline follows immediately as the trailing ? may in a contextual scan.

A history reference may be given without an event specification, i.e., **!$**. In this case, the previous command is being referenced unless a previous history reference occurred on the same line. In this case, the form repeats the previous reference. Thus, !?*foo*?^ **!$** gives the first and last arguments from the command matching ?*foo*?.

A special abbreviation of a history reference occurs when the first nonblank character of an input line is a ^. This is equivalent to **!:s**^, providing a convenient short-hand for substitutions on the text of the previous line. Thus ^*lb*^*lib* fixes the spelling of *lib* in the previous command. Finally, a history substitution may be surrounded with { and } if necessary to insulate it from the characters that follow. Thus, after **ls -ld ~paul**, **!{l}a** could be used to do **ls -ld ~paula**, while **!la** would look for a command starting with **la**.

Quotations With Single And Double Quotes

Quoting by ' ' and " " can prevent all or some remaining substitutions. Strings enclosed in ' ' are prevented from any further interpretation. Strings enclosed in " " may be expanded as described below.

In both cases, the resulting text becomes (all or part of) a single word; only in one special case (see **Command Substitution** below) does a " " quoted string yield parts of more than one word; ' ' quoted strings never do.

Alias Substitution

The shell maintains a list of aliases that can be established, displayed, and modified by the **alias** and **unalias** commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, the text that is the alias for that command is reread with the history mechanism available as though that command were the previous line input. The resulting words replace the command and argument list. If the history list is not referred to, the argument list is unchanged.

Thus, if the alias for "ls" is "ls -l" the command **ls /usr** would map to **ls -l /usr**; the argument list would not be disturbed. Similarly if the alias for "lookup" was "grep !^ /etc/passwd", **lookup bill** would map to **grep bill /etc/passwd**.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus, **alias print 'pr \!* | lpr'** can be done to make a command that **pr's** its arguments to the line printer.

Variable Substitution

The shell maintains a set of variables. Each variable has a list of zero or more words as a value. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the **set** and **unset** commands. Of the variables referred to by the shell, a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle that causes command input to be echoed. The setting of this variable results from the **-v** command line option.

Other operations treat variables numerically. The **@** command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed and before each command is executed, variable substitution is performed as keyed by **$** characters. This expansion can be prevented by preceding the **$** with a \ except within **" "**, where it always occurs, and within **' '**, where it never occurs. Strings quoted by **\` \`** are interpreted later (see **Command Substitution** below) so **$** substitution does not occur there until later, if at all. A **$** is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. Thus, the first (command) word to this point can generate more than one word. The first word becomes the command name, and the rest become arguments.

Unless enclosed in **" "** or given the **:q** modifier the results of variable substitution may eventually be command and file name substituted. Within **" "**, a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variable value separated by blanks. When the **:q** modifier is applied to a substitution, the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or file name substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, referring to a variable that is not set results in an error.

$*name* **${***name***}**	These are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters that would otherwise be part of it. Shell variables have names with up to 20 letters and digits starting with a letter. The underscore character is considered a letter. If *name* is not a shell variable but is set in the environment, then that value is returned. (But : modifiers and the other forms given below are not available in this case.)
$*name***[***selector***]** **${***name***[***selector***]}**	May be used to select only some of the words from the value of *name*. The selector is subjected to **$** substitution and may consist of a single number or two numbers separated by a -. The first word of a variable value is numbered 1. If the first number of a range is omitted, it defaults to 1. If the last member of a range is omitted it defaults, to **$#***name*. The selector ***** selects all words. An empty range is not an error if the second argument is omitted or is in range.
$#*name* **${#***name***}**	Gives the number of words in the variable. This is useful for later use in a **[***selector***]**.
$0	Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.
$*number* **${***number***}**	Equivalent to **$***argv***[***number***]**.
$*	Equivalent to **$***argv***[*]**.

The modifiers **:h**, **:t**, **:r**, **:q** and **:x** may be applied to the substitutions above as may **:gh**, **:gt** and **:gr**. If braces appear in the command form, the modifiers must appear within the braces. The current implementation allows only one : modifier for each **$** expansion.

The following substitutions may not be modified with : modifiers.

$?*name* **${?***name***}**	Substitutes the string **1** if name is set, **0** if it is not.
$?0	Substitutes **1** if the current input file name is known, **0** if it is not.

$$ Substitutes the (decimal) process number of the (parent) shell.

$< Substitutes a line from the standard input, with no further interpretation. It can be used to read from the keyboard in a shell script.

Command And File Name Substitution

The remaining substitutions, command and file name substitution, are applied selectively to the arguments of built-in commands. Thus, portions of expressions not evaluated are subject to these expansions. For commands not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution is indicated by a command enclosed in **\` \`**. The output from such a command is normally broken into separate words at blanks, tabs, and newlines, with null words discarded. This text then replaces the original string. Within **" "**, only newlines force new words; blanks and tabs are preserved.

In any case, the final newline does not force a new word. Thus, a command substitution can yield only part of a word even if the command outputs a complete line.

If a word contains *****, **?**, **[**, or **{** or begins with the character **~**, the word is a candidate for file name substitution, also known as "globbing". This word is then regarded as a pattern and replaced with an alphabetically-sorted list of file names that match the pattern. In a list of words specifying file name substitution, no pattern matching an existing file name results in as error, but it is not required for each pattern to match. Only the metacharacters *****, **?**, and **[** imply pattern matching. The characters **~** and **{** being more like abbreviations.

In matching file names, the character **.** at the beginning of a file name or immediately following a **/**, and the character **/** must be matched explicitly. The character ***** matches any character string, including the null string. The character **?** matches any single character. The sequence **[...]** matches any one of the characters enclosed. Within **[...]**, a pair of characters separated by **—** matches any character lexically between the two.

The character **~** at the beginning of a file name refers to home directories. Standing alone ("~"), it expands to the invokers home directory as reflected in the value of the variable *home*. When followed by a name consisting of letters, digits, and **—** characters, the shell searches for a user with that name and substitutes their home directory; thus, **~ken** might expand to **/usr/ken** and **~ken/chmach** to **/usr/ken/chmach**. If **~** is followed by a character other than a letter or **/** or does not appear at the beginning of a word, it is undisturbed.

The metanotation *a*{*b*,*c*,*d*}*e* is shorthand for *abe ace ade*. Left to right order is preserved, and results of matches are sorted separately at a low level to preserve this order. This construct may be nested. Thus,

~src/s1/{oldls,ls}.c expands to **/usr/src/s1/oldls.c /usr/src/s1/ls.c** whether or not these files exist with no chance of error if the home directory for **src** is **/usr/src**. Similarly, **../{memo,*box}** might expand to **../memo ../box ../mbox**. (Note that **memo** was not sorted with the results of matching ***box**.) As a special case **{**, **}**, and **{ }** pass undisturbed.

Input/Output

The standard input and output of a command may be redirected with the following syntax:

< *name* — Open file *name* (which is first variable, command and file name expanded) as the standard input.

<< *word* — Read the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, file name or command substitution, and each input line is compared to *word* before any substitutions are performed on this input line. Unless a quoting \, ", ', or ` appears in *word*, variable and command substitution is performed on the intervening lines, allowing \ to quote **$**, **\**, and **`**. Commands that are substituted have all blanks, tabs, and newlines preserved, except for the final newline, which is dropped. The resulting text is placed in an anonymous temporary file given to the command as standard input.

> *name*
>! *name*
>& *name*
>&! *name* — The file *name* is used as standard output. If the file does not exist, it is created; if the file exists, it is truncated and previous content is lost.

If the variable *noclobber* is set, the file must not exist or be a character special file (i.e., a terminal or **/dev/null**) or an error results. This helps prevent accidental destruction of files. In this case, the **!** forms can be used and suppress this check.

The forms involving **&** route the diagnostic output to the specified file and the standard output. *Name* is expanded as < input file names are.

>> *name*
>>& *name*
>>! *name*
>>&! *name* — Uses file *name* as standard output like >, but places output at the end of the file. If the variable *noclobber* is set, then it is an error for the file not to exist unless one of the **!** forms is given. It is otherwise similar to >.

A command receives the environment the shell was invoked in as modified by the input/output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands cannot access the text of the commands by default. Instead, they receive the original standard input of the shell. The **<<** mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input. Note that the default standard input for a command run detached is not modified to be the empty file **/dev/null**; rather, the standard input remains the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, the process will block and the user will be notified (see **Jobs** above).

Diagnostic output may be directed through a pipe with the standard output. To do so, the form **|&** rather than **|** should be used.

Expressions

A number of the built-in commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the **@**, **exit**, **if**, and **while** commands. The following operators are available:

```
|| && | ^ & == != =~ !~ <= >= < >
<< >> + - * / % ! ~ ( )
```

Here the precedence increases to the right, **==**, **!=**, **=~**, and **!~**; **<=**, **>=**, **<**, and **>**; **<<** and **>>**; **+** and **−**; and *****, **/**, and **%** being, in groups, at the same level. The **==**, **!=**, **=~**, and **!~** operators compare their arguments as strings; all others operate on numbers. The operators **=~** and **!~** are like **!=** and **==** except that the right-hand side is a *pattern* (containing, e.g., *****'s, ?'s, and instances of **[...]**) that the left hand operand is matched against. This reduces the need to use the **switch** statement in shell scripts when only pattern matching is needed.

Strings that begin with 0 are octal numbers. Null or missing arguments are 0. The result of all expressions are strings, which represent decimal numbers. No two components of an expression can appear in the same word. When adjacent to components of expressions syntactically significant to the parser (**&**, **|**, **(**, **)**, **<**, **>**) they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in **{** and **}** and file inquiries of the form *-l name*, where *l* is one of the following:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero size
f	plain file
d	directory

The specified *name* is command and file name expanded and then tested for the specified relationship to the real user. If the file does not exist or is inaccessible, all inquiries return false (0). Command executions succeed, returning true (1), if the command exits with status 0. Otherwise, they fail, returning false (0). If more detailed status information is required, the command should be executed outside of an expression and the variable *status* should be examined.

Control Flow

The shell contains a number of commands used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The **foreach**, **switch**, and **while** statements, and the **if-then-else** form of the **if** statement require the major keywords to appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers input whenever a loop is read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward **goto**'s will succeed on nonseekable inputs.)

Built-in Commands

Built-in commands are executed within the shell. If a built-in command occurs as any component of a pipeline except the last, it is executed in a subshell.

alias [*name* [*wordlist*]]
: If no arguments are given, prints all aliases. If just *name* is given, prints the alias for *name*. Otherwise, assigns the specified *wordlist* as the alias of *name*; *wordlist* is command and file name substituted. *Name* cannot be "alias" or "unalias".

alloc Shows the amount of dynamic memory acquired, broken into used and free memory. With any argument, shows the number of free and used blocks in each size category. The categories start at size 8 and double at each step. This command's output may vary across system types.

bg [**%***job* ...]
: Puts the current or specified jobs in the background, continuing them if they are stopped.

break Causes execution to resume after the **end** of the nearest enclosing **foreach** or **while**. The remaining commands on the current line are executed. Multilevel **breaks** are thus possible by writing them all on one line.

breaksw
: Causes a break from a **switch**, resuming after the **endsw**.

case *label:*
A *label* in a **switch** statement as discussed below.

cd [*name*]
chdir [*name*]
Change the shell's working directory to directory *name*. If no argument is given, change to the home directory. If *name* is not found as a subdirectory of the current directory (and does not begin with /, ./ or ../), each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with /, the *name* is tried to see if it is a directory.

continue
Continue execution of the nearest enclosing **while** or **foreach**. The remaining commands on the current line are executed.

default:
Labels the default case in a **switch** statement. The **default** should follow all **case** labels.

dirs Prints the directory stack. The top of the stack is at the left. The first directory in the stack is the current directory.

echo [**-n**] *wordlist*
The specified words are written to the shell's standard output, separated by spaces, and terminated with a newline unless the **-n** option is specified.

else
end
endif
endsw See the description of the **foreach**, **if**, **switch**, and **while** statements below.

eval *arg* ...
(As in *sh*(1).) The arguments are read as input to the shell and the resulting command(s) executed in the context of the current shell. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See *tset*(1) for an example of using **eval**.

exec *command*
The specified command is executed in place of the current shell.

exit [(*expr*)]
If *expr* is not given, the shell exits with the value of the *status* variable. Otherwise, the shell exits with the value of the specified *expr*.

fg [%*job* ...]
Brings the current or specified jobs to the foreground, continuing them if they are stopped.

foreach *name* (*wordlist*)

...

end The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching **end** are executed. (Both **foreach** and **end** must appear alone on separate lines.)

The built-in command **continue** may be used to continue the loop prematurely and the built-in command **break** may be used to terminate it prematurely. When this command is read from the terminal, the loop is read once, prompting with "?" before any statements in the loop are executed. If a mistake is made while typing in a loop at the terminal, it can be erased.

glob *wordlist*
Like **echo**, but no \ escapes are recognized and words are delimited by null characters in the output. Useful for programs that wish to use the shell to file name expand a list of words.

goto *word*
The specified *word* is file name and command expanded to yield a string of the form *label*. The shell rewinds its input as much as possible and searches for a line with the form *label*:, possibly preceded by blanks or tabs. Execution continues after the specified line.

history [**-rh**] [*n*]
Displays the history event list. If *n* is given, only the *n* most recent events are printed. The **-r** option reverses the order of printout so that the most recent is first, not the oldest. The **-h** option prints the history list without leading numbers. This will produce files suitable for sourcing using the **-h** option to **source**.

if (*expr*) *command*
If the specified expression evaluates true, the single *command* with arguments is executed. Variable substitution on *command* happens early, when it does for the rest of the **if** command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false and *command* is not executed. (This is a bug.)

if (*expr*) **then**

...

[**else if** (*expr2*) **then**]

...

[**else**]

...

endif If the specified *expr* is true, the commands to the first **else** are executed; otherwise, if *expr2* is true, the commands to the second **else** are executed, and so on. Any number of **else-if** pairs are possible; only one **endif** is needed. The **else** is optional. (The words **else** and **endif** must appear at the beginning of input lines; the **if** must appear alone on its input line or after an **else**.)

jobs [-l]
Lists the active jobs. With the -l option, it lists process IDs in addition to the normal information.

kill [-l] [-*sig*] [%*job* . . .] [*pid* . . .]
Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in **<signal.h>**, without the prefix "SIG"). The signal names are listed by **kill** -l. This command has no default. Therefore, using only **kill** will not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), the job or process will be sent a CONT (continue) signal as well.

limit [**-h**] [*resource*] [*maximum-use*]
Limits the consumption by the current process and each process it creates so that it does not individually exceed *maximum-use* on the specified *resource*. If no *maximum-use* is given, the current limit is printed; if no *resource* is given, all limitations are given. If the **-h** flag is given, the hard limits are used instead of the current limits. The hard limits impose a ceiling on the values of the current limits. Only the super-user may raise the hard limits, but a user may lower or raise the current limits within the legal range.

The *resource* that can currently be controlled is **filesize** (the largest single file which can be created).

The *maximum-use* may be given as a (floating-point or integer) number followed by a scale factor. The default scale is **k** or **kilobytes** (1024 bytes). A scale factor of **m** or **megabytes** may also be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

login Terminate a login shell, replacing it with an instance of **/bin/login**. This method for logging off is compatible with *sh*(1).

logout Terminate a login shell. Especially useful if *ignoreeof* is set.

nohup [*command*]
Without a *command*, **nohup** can be used in shell scripts to ignore hangups for the remainder of the script. Specifying a *command* runs the *command* with hangups ignored. All processes detached with **&** are effectively **nohuped**.

notify [%*job* . . .]
Causes the shell to notify the user asynchronously when the status of the current or specified jobs changes; normally notification is presented before a prompt. This is automatic if the shell variable *notify* is set.

onintr [-]
onintr [*label*]

Control the action of the shell on interrupts. With no options, **onintr** restores the default action of the shell on interrupts, which is to terminate shell scripts or to return to the terminal command input level. The form **onintr** - causes all interrupts to be ignored. The form **onintr** *label* causes the shell to execute a **goto** *label* when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running detached and interrupts are being ignored, all forms of **onintr** have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

popd [+*n*]

Pops the directory stack, returning to the new top directory. With an argument +*n*, this discards the *nth* entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

pushd [*name*]
pushd [+*n*]

With no arguments, **pushd** exchanges the top two elements of the directory stack. Given a *name* argument, **pushd** changes to the new directory (ala **cd**) and pushes the old current working directory (as in **csw**) on the directory stack. With a numeric argument, rotates the *nth* argument of the directory stack to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

rehash

Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while the user is logged in. This should only be necessary if commands are added to one of the user's directories or if a systems programmer changes the contents of one of the system directories.

repeat *count command*

The specified *command*, subject to the same restrictions as the *command* in the one-line **if** statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

set [*name*[[*index*]]=*word* ...]
set *name*=(*wordlist*) ...

With no options, shows the value of all shell variables. Variables that have more than a single word as a value print as a parenthesized word list. The form **set** *name* sets *name* to the null string. The form **set** *name*=*word* sets *name* to the single *word*. The form **set** *name*[*index*]=*word* sets the *index*th component of *name* to *word*; this component must exist. The form **set** *name*=(*wordlist*) sets *name* to the list of words in *wordlist*. The value is command and file name expanded.

These arguments may be repeated to set multiple values in a single **set** command. However, variable expansion happens for all arguments before any setting occurs.

setenv [*name* [*value*]]
With no options, **setenv** lists all current environment variables. When only *name* given, it is set to an empty string. If both *name* and *value* are given, the value of environment variable *name* is set to *value*, a single string. The most commonly-used environment variables USER, TERM, and PATH are automatically imported to and exported from the *csh* variables *user*, *term*, and *path*; **setenv** is not needed for these.

shift [*variable*]
The members of *argv* are shifted to the left, discarding *argv[1]*. *argv* must be set and have one word or more for a value. If *variable* is given, **shift** performs the same function on the specified *variable*.

source [**-h**] *name*
The shell reads commands from *name*. **Source** commands may be nested. If they are nested too deeply, the shell may run out of file descriptors. An error in a **source** at any level terminates all nested **source** commands. Normally, input during **source** commands is not placed on the history list; the **-h** option causes the commands to be placed in the history list without being executed.

stop [%*job* ...]
Stops the current or specified *job* executing in the background.

suspend
Causes the shell to stop as if it had been sent a stop signal with <CONTROL>-Z. This is most often used to stop shells started by *su*(1).

switch (*string*)
case *str1*:
...
breaksw
...
default:
...
breaksw
endsw Each case label is successively matched against the specified *string*, which is first command and file name expanded. The file metacharacters *, ? and [...] may be used in the **case** labels, which are variable expanded. If no labels match before a **default** label is found, execution begins after the **default** label. Each **case** label and the **default** label must appear at the beginning of a line. The command **breaksw** causes execution to continue after the **endsw**. Otherwise, control may fall through **case** labels and **default** labels as in C. If

no label matches and there is no **default**, execution continues after the **endsw**.

umask *value*
: The file creation mask is either displayed if no argument is given or set to the specified *value*. The mask is given in octal. Common values for the mask are 002, giving all access to the group and read and execute access to others, and 022, giving all access except no write access for users in the group or others.

unalias *pattern*
: All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by **unalias ***. It is not an error for nothing to be **unaliased.**

unhash
: Using the internal hash table to speed location of executed programs is disabled.

unlimit [**-h**] [*resource*]
: Removes the limitation on *resource*. If no *resource* is specified, then all *resource* limitations are removed. If **-h** is given, the corresponding hard limits are removed. Only the super-user may use this.

unset *pattern*
: All variables whose names match the specified *pattern* are removed. Thus, all variables are removed by **unset ***; this has noticeably distasteful side-effects. It is not an error for nothing to be **unset**.

unsetenv *pattern*
: Removes all variables whose names match the specified *pattern* from the environment. See the **setenv** command above.

wait Waits for all background jobs. If the shell is interactive, an interrupt can disrupt the **wait**. At this time the shell prints names and job numbers of all jobs known to be outstanding.

while (*expression*)
...
end While the specified *expression* evaluates to be nonzero, the commands between the **while** and the matching end are evaluated. **Break** and **continue** may be used to terminate or continue the loop prematurely. (The **while** and **end** must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the **foreach** statement if the input is a terminal.

%*job* Brings the specified *job* to the foreground.

%*job* **&**
: Continues the specified *job* in the background.

@ [*name*[[*index*]]=*expr*]
: With no options, prints the values of all shell variables. The form **@** *name*=*expr* sets the specified *name* to the value of *expr*. If the

expression contains <, >, &, or |, this part of the expression must be placed in ( ). The form @ *name*[*index*]=*expr* assigns the value of *expr* to the *index'th* argument of *name*. Both *name* and its *index*th component must exist.

The operators *=, +=, etcetera are available as they are in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* that would otherwise be single words.

Special postfix ++ and -- operators increment and decrement *name*, respectively, e.g., @ i++.

Predefined and Environment Variables

The following variables have special meaning to the shell. Of these, *argv*, *cwd*, *home*, *path*, *prompt*, *shell*, and *status*, the shell always sets. Except for *cwd* and *status*, this setting occurs only at initialization; these variables will not then be modified unless done explicitly by the user.

This shell copies the environment variable USER in the variable *user*, TERM in *term*, and HOME in *home* and copies these back in the environment when the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about setting it other than in the file **.cshrc**, as inferior *csh* processes will import *path*'s definition from the environment, and re-export it if it is changed.

argv	Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e., *$1* is replaced by *$argv[1]*, etc.
cdpath	Lists alternate directories searched to find subdirectories in **chdir** commands.
cwd	The full path name of the current directory.
echo	Set when the -**x** command line option is given. Echoes each command and its arguments just before it is executed. For nonbuilt-in commands, all expansions occur before echoing. Built-in commands are echoed before command and file name substitution, since these substitutions are then performed selectively.
filec	Enable file name completion.
histchars	Can be assigned a string value to change the characters used in history substitution. The first character of its value is the history substitution character, replacing the default character !. The second character replaces the character ^ in quick substitutions.
history	Can be assigned a numeric value to control the size of the history list. Any command referenced in this many events will not be discarded. Too large values of *history* may run the shell out of memory. The last executed command is always

saved on the history list.

home The invoker's home directory, initialized from the environment. The file name expansion of ~ refers to this variable.

ignoreeof Causes the shell to ignore end-of-file from input devices that are terminals. This prevents shells from accidentally being killed by <CONTROL>-D.

mail The files where the shell checks for mail. The check is done after each command completion that results in a prompt, if a specified interval has elapsed. The shell says "You have new mail" if the file exists with an access time not greater than its modify time.

If the first word of the value of *mail* is numeric, it specifies a mail checking interval, in seconds, that differs from the default, which is 10 minutes.

If multiple mail files are specified, the shell says "New mail in *name*" when mail is in the file *name*.

noclobber As described in the **Input/Output** section, output redirection is restricted to ensure that files are not accidentally destroyed and that **>>** redirections refer to existing files.

noglob If set, file name expansion is inhibited. This is most useful in shell scripts that do not deal with file names or after a list of file names has been obtained and further expansions are not desirable.

nonomatch If set, it is not an error for a file name expansion to not match any existing files; rather, the primitive pattern is returned. However, the primitive pattern still may not be malformed (e.g., **echo [** still gives an error).

notify If set, the shell notifies the user of job completions asynchronously. The default is to present job completions just before printing a prompt.

path Each word of the *path* variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable, only full path names will execute. The usual search path is **.**, **/bin**, and **/usr/bin**, but this may vary from system to system. For the super-user the default search path is **/etc**, **/bin**, and **/usr/bin**. A shell given neither the **-c** nor the **-t** option will normally hash the contents of the directories in the *path* variable after reading **.cshrc**, and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, the **rehash** command may need to be executed or the commands may not be found.

prompt	The string printed before each command is read from interactive terminal input. If a **!** appears in the string it will be replaced by the current event number unless a preceding \ is given. Default is "% ". ("# " for the super-user.)
savehist	The numeric value that controls the number of entries in the history list that are saved in **~/.history** when the user logs out. Any command referenced in this many events will be saved. During startup, the shell sources **~/.history** into the history list, enabling history to be saved across logins. Excessively large values of *savehist* will slow down the shell during startup.
shell	The file in which the shell resides. This is used in forking shells to interpret files with execute bits set, but cannot be executed by the system. (See the description of **Nonbuilt-in Command Execution** below.) It is initialized to the (system-dependent) home of the shell.
status	The status returned by the last command. If it terminated abnormally, 0200 is added to the status. Built-in commands that fail set *status* to 1. All other built-in commands set *status* to 0.
time	Controls automatic timing of commands. If set, any command that takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage, which is the ratio of user plus system times to real time to be printed when it terminates.
verbose	Set by the -**v** command line option, causes the words of each command to be printed after history substitution.

Nonbuilt-in Command Execution

When a command to be executed is not a built-in command, the shell attempts to execute the command using *execve*(2). Each word in the variable *path* names a directory from which the shell will attempt to execute the command. If it is given neither a -**c** nor a -**t** option, the shell will hash the names in these directories into an internal table so that it will only try an **exec** in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories is present in the search path. If this mechanism is turned off (through **unhash**), or if the shell has a -**c** or -**t** argument, and in any case for each directory component of *path* that does not begin with a **/**, the shell concatenates with the given command name. The concatenation forms a path name of a file that it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus, **(cd ; pwd) ; pwd** prints the *home* directory, but the current directory is unchanged (printing the current directory after the *home* directory). **cd ; pwd** changes the current directory to the *home* directory. Parenthesized commands are most often used to prevent **chdir** from affecting the current shell.

If the file has execute permission but is not an executable binary to the system, it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an **alias** for *shell* then the words of the alias will be prepended to the argument list to form the shell command. The first word of the **alias** should be the full path name of the shell (i.e., **$shell**). Note that this is a special, late occurring case of **alias** substitution and only allows words to be prepended to the argument list without modification.

Argument List Processing

If argument 0 to the shell is -, this is a login shell. The flag arguments are interpreted as follows:

-b This flag forces a "break" from option processing, causing any further shell arguments to be treated as nonoption arguments. The remaining arguments will not be interpreted as shell options. This may be used to pass options to a shell script without confusion or possible subterfuge. The shell will not run a set-user ID script without this option.

-c Commands are read from the (single) following argument that must be present. Any remaining arguments are placed in *argv*.

-e The shell exits if any invoked command terminates abnormally or yields a nonzero exit status.

-f The shell will start faster because it will neither search for nor execute commands from the file **.cshrc** in the invoker's home directory.

-i The shell is interactive and prompts for its top-level input even if it appears to not be a terminal. Shells are interactive without this option if their input and output are terminals.

-n Commands are parsed, but not executed. This aids in syntactic checking of shell scripts.

-s Command input is taken from the standard input.

-t A single line of input is read and executed. A \ may be used to escape the newline at the end of this line and continue onto another line.

-v Causes the *verbose* variable to be set. This variable causes command input to be echoed after history substitution.

-x Causes the *echo* variable to be set so that commands are echoed immediately before execution.

-V Causes the *verbose* variable to be set before **.cshrc** is executed.

-X Causes the *echo* variable to be set before **.cshrc** is executed.

After flag argument processing, if arguments remain but no **-c**, **-i**, **-s**, or **-t** options is given, the first argument is interpreted as the name of a command file to be executed. The shell opens this file and saves its name for possible resubstitution by $0. Since many systems use the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute such a "standard" shell if the first character of a script is not # (if the script does not start with a comment). Remaining arguments initialize the variable *argv*.

Signal Handling

The shell normally ignores *quit* signals. Jobs running detached (either by & or the **bg** or **%... &** commands) are immune to signals generated from the keyboard, including hangups. Other signals have the values the shell inherited from its parent. The shell's handling of interrupt and terminate signals in shell scripts can be controlled by **onintr**. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. Interrupts are never allowed when a login shell is reading the file **.logout**.

FILES

~/.cshrc	read at beginning of execution by each shell
~/.login	read by login shell, after **.cshrc** at login
~/.logout	read by login shell, at logout
/bin/sh	standard shell, for shell scripts not starting with a #
/tmp/sh*	temporary file for <<
/etc/passwd	source of home directories for ~*name*

SEE ALSO

signal(2), sigset(2), killpg(2B), a.out(4).
termio(7S) in the *CLIX System Administrator's Reference Manual.*
sh(1), access(2), execve(2), fork(2), pipe(2), ulimit(2), umask(2), wait(2) in the *UNIX System V Programmer's Reference Manual.*
environ(5) in the *UNIX System V System Administrator's Reference Manual.*

BUGS

When a command is restarted from a stop, the shell prints the directory it started in if it differs from the current directory. This can be misleading as the job may have changed directories internally.

Shell built-in functions cannot be stopped or restarted. Command sequences with the form *a* ; *b* ; *c* are also not handled gracefully when stopping is attempted. If *b* is suspended, the shell will then immediately execute *c*. This is especially noticeable if this expansion results from an **alias**. It suffices to place the sequence of commands in () to force it to a subshell, i.e., (*a* ; *b* ; *c*).

Tty output control after processes are started is primitive; a good virtual terminal interface with improved output control is needed. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops (prompted for by "?") are not placed in the history list. Control structure should be parsed rather than recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with **|**, and to be used with **&** and **;** metasyntax.

It should be possible to use the **:** modifiers on the output of command substitutions. All (at least more than one) **:** modifiers should be allowed for **$** substitutions.

Implementation of the **filec** facility is ugly and expensive.

CAVEATS

Words can be no longer than 1024 characters. The system limits argument lists to 12K characters. The number of arguments to a command that involves file name expansion is limited to 1/6 the number of characters allowed in an argument list. Command substitutions may substitute no more characters than those allowed in an argument list. To detect looping, the shell restricts the number of **alias** substitutions on a single line to 20.

NAME

cumail - DNP mail transport program

SYNOPSIS

cumail *mail-path* ...

DESCRIPTION

cumail is the Digital Network Protocol (DNP) electronic mail facility that allows mail to be exchanged between users on any host that supports the VAX/VMS mail protocol.

The format of *mail-path* for sending mail from *cumail* to DECnet node recipients is one of the following:

> *node_name*::*mail-address*
> *node_number*::*mail-address*
> *area_number.node_number*::*mail-address*
> *mail-address*@*node_name*.**CommUnity**

mail-address is a string that is handled by the remote node to determine the recipient. This address is passed to the remote node as a string and can cause further routing by the remote node if supported on that node.

cumail receives the body of incoming messages in either VAX/VMS variable record file format or Ultrix stream format and sends all outgoing mail in variable record format. *cumaild*(1M) is the server that receives requests from a *cumail* client.

cumail processes mail requests that contain DECnet-style addressing as part of a recipient address. *cumail* depends on *sendmail*(1) to route local mail or send requests to a remote recipient that does not have a DECnet address.

EXAMPLES

The following is an example of using *mail*(1):

```
mail mary vax::sally mike@node.uucp brian@vax.CommUnity
```

This command line uses other mailers for "mary" and "mike". It also calls *cumail* with the following command to send mail to "sally" and "brian":

```
cumail vax::sally vax::brian
```

SEE ALSO

cumaild(1).

NAME

dbg - symbolic debugger

SYNOPSIS

dbg [*option* ...] *objfil*

DESCRIPTION

dbg is an Intergraph-developed symbolic debugger supporting high-level language debugging for executables derived from C and FORTRAN source code. Executables derived from other languages can be examined and manipulated, but in a symbolic or absolute disassembly mode only. *dbg* features include built-in language expression parsing and evaluation for C and FORTRAN, a multiple screen window display mode, conditional breakpoints, hardware-assisted watchpoints, command-line recall, core-file debugging, and online help.

Objfil is assumed to be an executable program file. If source-line debugging will be performed, one or more routines in *objfil* should be compiled with the **-g** (debug) compiler option.

When invoked, *dbg* will examine the symbol table of *objfil* and attempt to create a process from the executable. Assuming the process creation is successful, *dbg* will then open the address space of the process through the **proc** file system. If the **-c** (core file debugging) option is specified, *dbg* will set the process state according to the contents of the *core*(4) file. A set of commands may be read and executed from an input file. Once these steps have been completed, *dbg* is ready for command input.

Command interpretation in *dbg* is table-driven and supports abbreviation of command names and options. Options are always introduced by the slash (/) character. Argument interpretation is handled primarily by the language expression parsers and a common expression evaluator.

Variables and expressions used during a debugging session are interpreted according to the current high-level source language. (The current source language setting is controlled through the **language** command described below.) Syntax errors similar to those displayed by the respective language compilers are generated for improper variable references and expressions.

High-level language expressions are the primary vehicles for formulating requests and initiating action in *dbg*. Expressions are used to perform common operations such as examining variables, depositing data in variables, and controlling the flow of debugger commands. The **evaluate** command is provided for parsing and evaluating language expressions.

In addition to evaluating source language expressions involving variables declared in the process being debugged, users can, through the **declare** command, create instances of local debugger variables. These variables, like expressions, are interpreted in the context of the current high-level source language. Once defined, debugger variables may be used in expressions just as if they were variables contained within the process with one exception:

local debugger variables cannot reference addresses in the address space of the process, nor can process variables reference addresses of debugger variables. A fixed precedence order resolves name conflicts between process variables and local variables. Mechanisms for overriding precedence are provided. Local debug variable definitions may be removed with the **undeclare** command.

dbg supports iteration and flow control commands in the form of built-in **while** and **if** statements. These commands rely on the expression parsing and evaluation for the *condition* portion of the command. The *action* portion of the command can be a combination of debugger commands and expressions.

The command-line *options* available are as follows:

-c *corfil*	Set process to the state defined in *corfil*, which should be a *core*(4) file. Open file information, shared memory, and semaphores are not recorded.
-p *path*[*:path* ...]	Define the search path for source files.
-w	Display any caveat about the symbol table when the process is loaded for debugging (suppressed by default).
-e	Make **evaluate** the default *dbg* command.
-h *histfile*	Use *histfile* as the history file for this debug session. The default is **$HOME/.dbg_history**.
-P *prompt*	Define the prompt for *dbg*.

All command-line options must precede the name of the process file to be debugged. Arguments to the debug process must be positioned after the process file name.

Commands

The following conventions are used in describing commands:

1) Permissible abbreviations for commands and options are indicated with bolded characters.

2) *Cmd-list* may be a single command or several commands enclosed in braces { } and separated by semicolons.

break
Display the breakpoints.

break *bp*[,*bp* ...] *cmd-list*
Set breakpoints.

break/*count* *bp*[,*bp* ...] *cmd-list*
Break on *count*th occurrence.

break/delete
Delete breakpoints and confirm before deletion.

break/delete
Delete breakpoints.

break/delete/all
Delete all known breakpoints.

break/quiet *bp* [,*bp* ...]
Do not display stop information for this breakpoint.

break/return *bp* [,*bp* ...]
Establish a breakpoint at the last instruction of the function identified by *bp*. The return value of the function will be displayed when the break occurs.

declare
Display local variable declarations.

declare *declaration* [,*declaration* ...]
Declare the specified local debug variables. *Declaration* may be any valid C or FORTRAN declaration statement. The source language used to parse *declaration* is established by the **language** command.

evaluate
Display the succeeding data item using the type and format of the preceding *evaluate* command. The starting address is incremented according to the data type.

evaluate *expression* [,*expression* ...]
Evaluate *expression*.

evaluate/*type*: [*count*] *special-expression*
Evaluate the contents of the address yielded from the *special-expression* as *count* items of type *type*.

evaluate/*format expression*
Evaluate the *expression* and display as *format*.

evaluate/*special expression*
Evaluate the *expression* using the *special* method.

Expression may be any valid C or FORTRAN expression. *Special-expression* may be an integer constant, function name, or an expression yielding an *lvalue*. *Type*, *format*, and *special* may be combined to control the action of **evaluate**.

Type applies only to expression results that possess addresses (*lvalues*). Valid *type* values are **character**, **double**, **float**, **instruction**, **integer**, **long**, and **short**. The following rules apply to the default *type* used for the display when addresses are specified as integer constants:

/instruction:1	if the address is a text address
/char:1	if the address is not on an even-byte boundary
/short:1	if the address is not aligned on a four-byte boundary

/int:1	otherwise

Format controls the display of expression evaluation results. Valid values for *format* are **decimal**, **hexadecimal**, **octal**, **unsigned**, and **x** (same as **hexadecimal**). The default *format* is **decimal**.

Special options modify the behavior of **evaluate** as indicated:

address	Examine the address, not the contents, of the given symbol.
debug	Search only variables local to the debugger.
environ	Change the search algorithm to check environment variables first.
follow	Modify the display algorithm to follow pointers to structures and unions.
global	Change the search algorithm to check global variables first.
quiet	Suppress output generated by the evaluated command.
register	Change the search algorithm to check register names first. If no expression is given, all registers are displayed.
static	Change the search algorithm to check static variables first.
string	Display ASCII characters until the first null byte is encountered.
symbolic	Display the closest symbol plus offset (if any) for text and data values.
type	Display the type of the variable or expression. (The expression is not evaluated.)
value	Echo back integer constants. (This option is useful to display a constant using a different format.)

The default search order for looking up symbol names in expressions is as follows: process local variables, process static variables, process global variables, debugger variables, environment variables, and process registers.

find Repeat the last search.

find *pattern*
Search for a pattern in the currently scoped source file. *Pattern* is a nonempty sequence of characters delimited by any character not in the sequence. The search begins at the current line.

find/forward
Repeat last search forward.

find/forward *pattern*
Search forward for a pattern.

find/backward
Repeat last search backward.

find/backward *pattern*
: Search backward for a pattern.

find/regular *pattern*
: Interpret *pattern* as a regular expression. The regular expression follows the conventions described in *ed*(1).

go
: Resume executing the process after *dbg* received control due to a **break**, **watch**, **signal**, or user intervention.

go *bp* [**,** *bp* ...]
: Set temporary breakpoints and continue processing.

go/pass
: Continue processing; pass any pending signals to the process.

go/pass *bp* [**,** *bp* ...]
: Pass signals, set temporary breakpoints, and continue processing.

go/delete *bp* [**,** *bp* ...]
: Delete specified breakpoints and continue.

go/delete/all
: Delete all breakpoints and continue.

go/return
: Continue processing; stop at the end of the current function and display its return value.

go/return *function*
: Continue processing; stop at the end of *function* and display its return value.

help [*dbg-cmd* [**/***dbg-option* ...]
: Access the *dbg* online help facility.

if (*condition*) *cmd-list1*
[**else** *cmd-list2*]
: If *condition* is true, execute *cmd-list*. Otherwise, execute *cmd-list2*.

  The **if** statement is most useful when used as part of a command list on statements such as **break**, **watch**, and **step**.

kill
: Terminate the currently active process. The currently active process can also be killed through the **run** command by either restarting the process or defining a new one.

language
: Display the language setting. The language setting controls interpretation of language expressions and declarations.

language/c
: Set the default language to C.

language/fortran
: Set the default language to FORTRAN.

language/**macro**
: Set the default language to the machine language. This affects the **step** command by altering the default action from **/line** to **/instruction** and causes symbol lookup to check for matches on register names before process or debug variables.

process
: Display information on all currently active *dbg* processes.

process/**attach** *exefil*
: Attach to the existing process *exefil*.

process/**attach** *pid*
: Attach to the existing process whose ID is *pid*.

process/create *exefil* [*args*]
: Create a new process under the control of *dbg*.

process/**kill**
: Kill all processes using the confirmation mode.

process/**kill** *exefil*
: Kill the process *exefil*.

process/**kill** *pid*
: Kill the process with process ID *pid*.

process/**kill**/**all**
: Kill all processes.

quit Terminate the debugging session.

redirect *file-name*
: Redirect the output of *dbg* to *file-name*.

redirect/**append** *file-name*.
: Append the output of *dbg* to *file-name*.

redirect/off
: Redirect the output of *dbg* back to **stdout**.

run Start or restart the process. Any previously defined arguments are recalled.

run *args*
: Start or restart the process, passing the argument list specified in *args* to it.

run *args* >*file-name*
: Start or restart the process, passing the argument list specified in *args* to it and redirecting the output to *file-name*. Full shell I/O redirection syntax is supported.

run/clear
: Start or restart the process, clearing any arguments previously passed.

run/clear *args*
: Start or restart the process, passing *args* to it.

run/recall
Start or restart the process, recalling any previously defined arguments.

run/recall *args*
Start or restart the process, appending *args* to any previously defined arguments.

run/new *process-file*
Start the process whose .text and .data reside in *process-file*.

run/new *process-file args*
Start the process whose .text and .data reside in *process-file*. The argument list specified in *args* is passed to the new process.

scope
Display the current scope setting. The current scope defines source lines available for viewing with the **type** command and for searching with the **find** command. When control is returned to *dbg* due to a breakpoint or watchpoint, the scope is set to the function containing the current program counter (PC).

scope *function-name*
Set **scope** to the specified function.

scope *file-name*
Set **scope** to the specified file.

scope *"file-name"function*
Set **scope** to *function-name* in *file-name*.

screen
Enter screen window display mode. In this mode the screen is divided into three windows, one for source display, one for process and debugger output, and one for command entry. The source display window contents are automatically updated based on the current **scope** setting. The process/debugger output display window captures all information displayed on the standard output and error devices by *dbg* and the process being debugged. The source display window contents may also be altered through the **type** and **find** commands. <CONTROL>-W will switch between screen display windows. <CONTROL>-P and <CONTROL>-N will scroll up and down, respectively, in the current screen display window.

screen/assembly
Display, in addition to the standard windows associated with the **screen** command, a disassembly window. The contents of this window are updated based on the value of the PC.

screen/off
Terminate the screen window display mode.

signal
Display all signal settings. This command allows individual signals affecting the process to be ignored by the process and/or by *dbg*, set so

that they return control to *dbg* (the default), or set so that they are passed to the process without causing *dbg* to regain control.

signal *signal*[,*signal* ...]
: Display settings of the listed signals.

signal/stop
: Display all signals set to **stop**.

signal/stop *signal*[,*signal* ...]
: Specify signals that, when caught, cause processing to stop and control to be passed to *dbg*.

signal/go
: Display all signals set to be ignored by *dbg* and the process.

signal/go *signal*[,*signal* ...]
: Specify signals to be ignored by both *dbg* and the process being debugged.

signal/go/pass
: Display all signals set to be ignored by *dbg* but passed to the process.

signal/go/pass *signal*[,*signal* ...]
: Specify signals to be ignored by *dbg* but passed to the process.

Signals can be specified by name or number. A range of signals can be specified by separating signal numbers with a colon (:). Names cannot be used in a range specification.

source
: Display the default source path. This command expands or restricts the source file directories searched for high-level language source files.

source ""
: Delete the current source definition.

source "*path-name*[:*path-name* ...]"
: Set the default source path.

source/append "*path-name*[:*path-name* ...]"
: Append given path names to the default source path.

stack
: Display a default number of frames.

stack/*count*
: Display *count* frames.

stack/all
: Display all frames.

stack/default
: Show the default **stack** command. Specified with other options, this command establishes those options as the default.

step Step instructions or source lines.

step/*count*
: Step *count* instructions or source lines.

step/line
: Step to the next source line.

step/instruction
: Step to the next instruction.

step/over
: Step over any function call to the next instruction or source line.

step/into
: Step into any function call to the next instruction of source line.

step/quiet
: Do not display source and/or instruction lines when stepping.

step/verbose
: Display source and/or instruction lines when stepping. This option can be used to temporarily override the **/quiet** option.

step/default
: Display the default **step** command. Specified with other options, this command will establish those options as the default. No stepping occurs.

type Display the next source line. This assumes that the current source line is defined. The current source line is defined and/or altered by the **type**, **find**, **scope**, and **step** commands and also as a result of interruption of process execution. The current source line will not be defined or altered if the process being debugged was not compiled with the **-g** (debug) switch. All forms of the **type** command will alter the current source line.

type/pc
: Display the source line representing the current PC.

type .
: Display the current source line. The current source line may differ from the source line representing the current PC due to prior use of the **scope**, **type**, or **find** command.

type *number*
: Display the specified source line.

type *number:number*
: Display a range of source lines. If the first number is omitted, it defaults to line number 1. The second number defaults to the last known line within the file. **.** can be used in either position.

type/*count*
: Display the next *count* source lines.

type/-*count*
: Display the previous *count* source lines. The current source line is set

to current − *count*.

undeclare
Undeclare local debug variables using the confirmation mode.

undeclare *variable*[**,***variable* ...]
Undeclare the specified local debug variables.

undeclare/all
Undeclare all local debug variables.

watch
Display the current watchpoint settings.

watch *wp*[**,***wp* ...] *cmd-list*
Set watchpoints.

watch/*count* *wp*[**,***wp* ...] *cmd-list*
Set watchpoints to stop on the *count*th modification of the memory associated with the respective *wp*.

watch/*type*[**:***count*] *wp*[**,***wp* ...] *cmd-list*
Watch locations as *count* number of elements of data type *type*.

watch/delete
Delete watchpoints using the confirmation mode.

watch/delete *wp*[**,***wp* ...]
Delete watchpoints.

watch/delete/all
Delete all watchpoints.

watch/quiet
Do not display stop information about the specified watchpoints.

while (*condition* **)** *cmd-list*
While *condition* is true, execute the commands in *cmd-list*.

!*command*
Execute *command* in the default shell.

!<RETURN>
Escape to the default shell.

The default shell is defined through the environment variable **SHELL**. *sh*(1) is used if this definition does not exist. To pass a semicolon or a right brace (}) to the shell so it is not seen as a *dbg* command separator, escape it by prefixing it with a backslash (\).

Miscellaneous Features

Command-line recall and editing features are as follows:

<RETURN>	Recall and execute the most recent command.
<CONTROL>-A	Go to the beginning of the line.
<CONTROL>-E	Go to the end of the line.

<CONTROL>-D	Delete the character the cursor is on.
<CONTROL>-P **<UP-ARROW>**	Recall the previous command or scroll the window contents in screen display mode.
<CONTROL>-K	Delete all characters to the right of the cursor.
<CONTROL>-N **<DOWN-ARROW>**	Recall the next command or scroll the window contents in screen display mode.
<CONTROL>-B **<LEFT-ARROW>**	Move the cursor to the left one position.
<CONTROL>-F **<RIGHT-ARROW>**	Move the cursor to the right one position.
<CONTROL>-W	In screen mode, switch to the next screen display window. The windows are visited in top-to-bottom order.
<CONTROL>-V	In screen mode, move the cursor down one page.
<ESC>-V	In screen mode, move the cursor up one page.
<DELETE>	Delete the character to the left of the cursor.
<KILL>	Delete the entire line.
<EOF> **<RETURN>** **<LINE FEED>**	Designate the end of a command. *dbg* then executes the line contents.

Environment Variables

DBGHISTSIZ defines the maximum number of lines written to the *dbg* history file. If this variable is not set, 50 is the maximum.

EXAMPLES

break *main*
Set a breakpoint at function *main.*

break @20
Set a breakpoint at line 20 of the current file.

break @"*input.c***"20**
Set a breakpoint at line 20 of the file *input.c.*

break/delete *main*
Delete the breakpoint set at function **main.**

break/10 *printf*
Set a breakpoint at function **printf** and return control to *dbg* every 10th time *printf* is called.

break/return *get_token* **{ eval/str** *token***; go}**
Print the return value of *get_token* and display *token* each time the function *get_token* returns.

break *read_file* **if** (*file_number* **!= 5) go**
: Break on function *read_file* if the variable *file_number* equals 5; otherwise, continue execution. The expressions in the condition and action portions of the **if** statement are executed when the break occurs and in the scope of the function where the break occurs.

evaluate/hex *i, j, k*
: Display the values of variables *i*, *j*, and *k* in hexadecimal.

evaluate/double:10 *dbl_ptr*
: Display the contents of the 10 double precision values beginning at the address contained in the variable *dbl_ptr*.

e/addr *i*
: Display the address of the variable *i*.

e/hex *stat.st_dev* **>> 8 & 0xff**
: Examine the low-order byte of the *st_dev* field of the *stat* structure.

e/reg f0 - (double)1.234
: Display the result of subtracting 1.234 from the floating point register **f0**.

find */if (i ==/*
: Search for the string *if (i ==* in the current file beginning at the current source line.

find/back *'now is the time'*
: Search backward from the current line in the current file for the string *now is the time*.

find/pat *a*[*bd*]*
: Search for a string beginning with *ab* or *ad* in the current file.

if (*i* **==** *j* **+ 20) go**
else { break *func* **; go }**
: Continue execution if variable *i* exceeds variable *j* by 20. Otherwise, set a breakpoint at function *func* and then continue execution.

run *a.out* **>***/dev/ttx01* **2>&1**
: Start or restart *a.out* with standard output and standard error redirected to */dev/ttx01*.

signal/go sighup
: Ignore SIGHUP in both *dbg* and the process.

signal/go/pass 10
: If signal number 10 is received by the process, pass it to the process without returning control to *dbg*.

step/def/sou/into/line
: Establish the default step action to be step by source line and step into called functions.

watch *x_pos*
: Watch the address range associated with variable *x_pos* and break

when a write to this range occurs.

watch/double *i*
: Watch the eight-byte address range beginning at the address of *i* and break when a write to this range occurs.

watch/char:20 *a* **if (pc >** *main* **&& pc <** *func1***) go**
: Watch the 20-byte address range beginning at the address associated with variable *a*. Break if the PC register is not in the address range associated with *main*. This effectively watches for writes to the array *a* that occur outside of the function *main*.

while (*i* **< 1000) step**
: Step until the variable *i* equals or exceeds 1000.

declare short *s
eval s = (short *)pc
while ((*s & 0xff00) != 0x4500) { step/instr ; eval s = (short *) pc }
: Step from the current PC in the process until a *call* instruction is encountered. (0x4500 is the opcode for a *call* instruction.) When a *call* instruction is found, control will be returned to *dbg*. This example also illustrates the use of debugger variables.

$ dbg a.out <*dbgcmds*
: Read commands from file *dbgcmds*. Input will revert to **/dev/tty** when EOF is encountered in *dbgcmds*.

FILES

$HOME/.dbghistory	dbg command history file
/usr/lib/dbg.hlp	online help file

SEE ALSO

cc(1), f77(1), a.out(4).
proc(7S) in the *CLIX System Administrator's Reference Manual.*
sh(1) in the *UNIX System V User's Reference Manual.*
syms(1) in the *UNIX System V Programmer's Reference Manual.*
"PROC Debugging Tutorial" in the *CLIX System Guide.*

NAME

dls - list contents of MS-DOS directory

SYNOPSIS

dls [-adflmrtFR] *name* ...

DESCRIPTION

dls lists the directory contents for MS-DOS directories. (*dls* performs the same function for MS-DOS directories as *ls*(1) does for CLIX directories.) If no directories are named on the command line, the root directory of drive **a:** is assumed.

The long form of the directory listing supplies the file name; the read, hidden, system, volume, directory, and archive bits; the date and time of the last file modification; the beginning File Allocation Table (FAT) entry of the file; and the total size (in bytes) of the file.

The options supported are as follows:

-a	List all entries, including hidden files, system files, ., and ..
-d	For each directory argument, list only its name, not its contents.
-f	Force each argument to be interpreted as a directory and list the name found in each entry. This option turns off **-l**, **-t**, **-s**, and **-r** and turns on **-a**.
-l	List in long format.
-m	Force stream output format.
-r	Reverse the sort order to get reverse alphabetic (default) or oldest first (if the **-t** option is specified).
-t	Sort by time modified (latest first) instead of by name.
-F	Mark directories with a trailing **/**.
-R	Recursively list subdirectories encountered.

Drive **a:** (the floppy drive), **b:** (the external floppy drive for systems with two floppy drives), or **c:** (the DOS partition of the hard disk) may be accessed with this program. If no drive is specified, **a:** (the floppy drive) is assumed.

EXAMPLES

```
dls                     (lists contents of a:\)
dls -al a:\foo
dls c:
```

FILES

/dev/dsk/f1	default floppy device
/dev/dsk/ufloppy	3½ inch floppy driver
/dev/dsk/floppy	5¼ inch floppy driver
/dev/dsk/s0u0p9.0	DOS partition

SEE ALSO

dtu(1).

NAME
domname - set or display name of current YP domain

SYNOPSIS
domname [*nameofdomain*]

DESCRIPTION
Without an argument, *domname* displays the name of the current domain. Only the super-user can set the domain name by giving an argument. Currently, domains are only used by the Yellow Pages (YP) to refer collectively to a group of hosts.

FILES
/etc/domainname	used to hold name of YP domain

SEE ALSO
ypinit(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

dtu, utd - copy between MS-DOS and CLIX

SYNOPSIS

dtu [**-p**] *file1 file2*
dtu [**-p**] *file ... directory*
dtu [**-p**] *file ...* >*file*

utd [**-p**] *file1 file2*
utd [**-p**] *file ... directory*
utd [**-p**] <*file file2*

DESCRIPTION

dtu copies files from MS-DOS to CLIX, and *utd* copies files from CLIX to MS-DOS.

Not all CLIX file names are legal under MS-DOS. An MS-DOS file name consists of eight or fewer characters and an extension of three or fewer characters. The following characters are illegal in MS-DOS file names:

? . , ; : = * / \ + " < >

If necessary (when using *utd* to copy several CLIX files to an MS-DOS directory), *utd* forms legal MS-DOS file names from CLIX file names. It does so by truncating any names or extensions that are too long and changing any illegal characters to **@**.

Normally, *utd* and *dtu* assume that text files are being copied and adjust for the difference in end-of-line and end-of-file conventions between the two systems. The **-p** flag will override this feature and cause the files to be transferred with no interpretation.

Drive **a:** (the floppy drive), **b:** (the external floppy drive for systems with two floppy drives), or **c:** (the DOS partition of the hard disk) may be accessed with this program. If no drive is specified, **a:** (the floppy drive) is assumed.

EXAMPLES

```
utd *.c a:\csrc
dtu *.h | pg
dtu "a:\*.*" .    (copy root directory of drive a: to current directory)
dtu -p a:command.com binfile
```

FILES

/dev/dsk/f1	default floppy device
/dev/dsk/floppy	5¼ inch floppy driver
/dev/dsk/ufloppy	3½ inch floppy driver
/dev/dsk/s0u0p9.0	DOS partition

SEE ALSO

dtu(1).

NOTES

MS-DOS path names that contain wildcards should be enclosed in quotation marks to prevent the shell from interpreting them.

NAME

efl - Extended FORTRAN Language

SYNOPSIS

efl [*option* ...] [*file* ...]

DESCRIPTION

efl compiles a program written in the EFL language into clean FORTRAN on the standard output. *efl* provides the C-like control constructs of *ratfor*(1):

statement grouping with braces.

decision-making:
if, **if-else**, and **select-case** (also known as **switch-case**)
while, **for**, FORTRAN **do**, **repeat**, and **repeat** ... **until** loops
multilevel **break** and **next**

EFL has C-like data structures, i.e.:

```
struct {
        integer flags(3)
        character(8) name
        long real coords(2)
} table(100)
```

The language offers generic functions, assignment operators (**+=**, **&=**, etc.), and sequentially evaluated logical operators (**&&** and **||**). It has a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) })
```

EFL also provides some syntactic "sugar":

free-form input:
multiple statements per line; automatic continuation; statement label names (not just numbers)

comments:
this is a comment

translation of relational and logical operators:
>, **>=**, **&**, etc., become **.GT.**, **.GE.**, **.AND.**, etc.

return expression to caller from function:
return (*expression*)

defines:
define *name replacement*

includes:
include *file*

efl understands several *option* arguments: **-w** suppresses warning messages, **-#** suppresses comments in the generated program, and the default *option* **-C** includes comments in the generated program.

An argument with an embedded = (equal sign) sets an EFL *option* as if it had appeared in an *option* statement at the start of the program. A set of defaults for a particular target machine may be selected by one of the choices: **system=unix**, **system=gcos**, or **system=cray**. The default setting of the system *option* is the same as the machine the compiler is running on.

Other specific *options* determine the style of input/output, error handling, continuation conventions, the number of characters packed per word, and default formats.

efl is best used with *f77*(1).

SEE ALSO

cc(1), f77(1), ratfor(1).

NAME

errors - error logging report generator

SYNOPSIS

errors [**-hnsrb**] [**-f** *file*] [**-z** "*time*"] [**-t** "*time*"] [**-i** *types*] [**-e** *types*]

DESCRIPTION

errors generates a report from an error log file. The report is sent to standard output. If no options are used, *errors* reports all entries in **/usr/adm/errlog** on the current system. Available options are as follows:

-h	Display a help screen.
-n	Report errors for a system other than the current system. *errors* will prompt for the network address, user name, and password. The error logging file for that system will be copied to **/usr/tmp/errlog** on the current system. This option will not work with **-f** or **-r**.
-s	Report the number of errors per device and per error type on a system. This option can be used with options **-n**, **-t**, **-z**, and **-f**.
-r	Instruct the error daemon, *errord*(1M), to send error messages to the error log file and to *errors* for immediate display. This option can be used with options **-i**, **-e**, and **-b**.
-b	Give an abridged version of the error logging report. This option cannot be used with **-s**.
-f *file*	Specify the log file to be used. The default is **/usr/adm/errlog**. This option will not work with **-n** or **-r**.
-t "*time*"	Specify the date and time to start the report. *Time* must be in quotation marks. This option will not work with **-r**. These are examples of valid times: "yesterday 13:34" "29-feb 1988 12:01" "12/25/88 10:30"
-z "*time*"	Specify the date and time to end the report. See examples above. This option will not work with **-r**.
-i *types*	Include only the error types specified by *types* in the report. Valid *types* are **device**, **user**, **panic**, **memory**, **slave**, **disk**, **tape**, **floppy**, **ascn**, **scan**, **parallel**, **digitizer**, **timeout**, **security**, **stray**, **optic**, **soft**, **retry**, and **hard**. If multiple *types* are specified, they must be separated by commas and/or spaces; if spaces are used, the entire *types* string must be enclosed in quota-

tion marks. This option will not work with **-e** or **-s**.

-e *types* Exclude only the error types specified by *types* from the report. Valid *types* are **device**, **user**, **panic**, **memory**, **slave**, **disk**, **tape**, **floppy**, **asycn**, **scan**, **parallel**, **digitizer**, **timeout**, **security**, **stray**, **optic**, **soft**, **retry**, and **hard**. This option will not work with **-i** or **-s**.

EXAMPLES

The following command will report all errors in the error log file:

```
errors
```

The following will prompt the user for the system to connect to and a username/password combination to use. That system's error log file is placed on the current system in **/usr/tmp/errlog**.

```
errors -n -t "yesterday 12:00" -i disk,memory
```

FILES

/usr/adm/errlog	system error log file
/usr/tmp/errlog	temporary error log (for errors from another system)

SEE ALSO

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

NAME

f77 - FORTRAN compiler

SYNOPSIS

f77 [*option* ...] *file* ...

DESCRIPTION

The *f77* command controls the compilation and link editing of FORTRAN and other source programs. The compilation process is divided into several passes. Each pass is invoked with appropriate arguments and options.

f77 uses the high-performance CLIPPER FORTRAN compiler developed by Green Hills Software, Inc. under Intergraph Corporation contract. The CLIPPER FORTRAN compiler has been designed to improve general code performance based on selectable optimizations.

Each command line argument represents an *option* or a *file* name. A large number of options (discussed below), and seven types of file name arguments are understood. Any file name or option not recognized are passed to the link editor.

The *file* arguments are processed in left to right order as they appear on the command line. The generated object files are passed on to the link edit pass in the same order.

Compilation Phases

The compilation phases and their names are largely historic. Each phase is approximately implemented as a single command. There are a number of options that control the invocation of each phase. Such options use key letters to indicate a particular phase.

The phases and their key letters are:

p The C preprocessor phase. This phase processes the preprocessor directives in a source file. Preprocessor directives are given on lines whose first character is the # symbol. The preprocessor implements file inclusion, conditional code inclusion, macro definition, and macro expansion (see *cpp*(1)).

0 (zero) The C source analysis phase. This phase analyzes the (preprocessed) source file according to the rules of the C language proper. Syntax and semantic errors are detected here. Typically, an internal or intermediate representation of the source file is built.

1 (one) The Fortran source analysis phase. This phase analyzes the source file according to the rules of the Fortran language proper. Syntax and semantic errors are detected here. Typically, an internal or intermediate representation of the source file is built.

a The assembler phase. The assembler phase translates the assembler code into an object (or binary) file. See *as*(1), the "Assembler" section of the "Technical Programming Tutorial" in the *CLIX System Guide*, and the *CLIPPER User's Manual.*

l The link edit phase. Startoff routines, generated objects, and standard libraries are linked together into an image file (see *ld*(1)).

m The macro preprocessing phase. The macro preprocessing phase expands *m4*(1) macros into the appropriate character sequences, (see *m4*(1)).

e The *efl*(1) phase. This phase preprocesses *efl*(1) commands and constructs into the appropriate Fortran source, (see *efl*(1)).

r The *ratfor*(1) phase. This phase preprocesses *ratfor*(1) commands and constructs into the appropriate Fortran source, (see *ratfor*(1)).

c The *cc*(1) phase. This phase invokes the *cc*(1) command on the indicated C source files, (see *cc*(1)).

The CLIPPER Fortran compiler implements the source analysis, and code generation phases in one program (**/lib/fcom**). For the options that take a phase key letter, **1** indicates this program.

The assembler (**/bin/as**) and link editor (**/bin/ld**) implement the assembler and link editor phases, respectively.

Each input file is processed by each phase in sequence. If an error occurs in a phase, further processing of the input file that contained the error is abandoned. (The assembler will not be invoked if a compiler error occurred). Any remaining input files are compiled (or assembled), but the link edit phase is not performed.

File Names

f77 recognizes seven types of file name arguments. Based on the suffix of each input file, *f77* selects the various preprocessors and compilers used to process the file. The output of each pass is a file whose suffix indicates the type of result. The suffixes *f77* recognizes or generates are listed below approximately in the order that the passes are performed.

.e EFL source file. The source file is translated using an EFL preprocessor resulting in a **.f** source file that is then processed as described under **.f** below. **.e** files may be optionally preprocessed by the *m4*(1) macro-processor before they are translated by the EFL preprocessor (see *efl*(1)).

.r RATFOR source file. The source file is translated using a RATFOR preprocessor resulting in a **.f** source file, that is then processed as described under **.f** below. **.r** files may be optionally preprocessed by the *m4*(1) macro-processor before they are translated by the RATFOR preprocessor (see *ratfor*(1)).

.F FORTRAN source file. The source file is compiled using the CLIPPER FORTRAN compiler. **.F** files may contain C preprocessor directives (i.e., **#define**) handled by the built-in CLIPPER FORTRAN C preprocessor. The compiler generates a **.s** file that is processed as described under **.s** below.

.f FORTRAN source file. The source file is compiled using the CLIPPER FORTRAN compiler. The compiler generates a **.s** file that is processed as described under **.s** below.

.c C source file. The source file is compiled using the *cc*(1) command resulting in a **.o** file (see *cc*(1)).

.s (Command Argument) Assembler source file. Each **.s** file given as a command argument is processed by the *cc*(1) command resulting in a **.o** file (see *cc*(1)).

.s (Generated) Assembler source file. Each assembler source file generated by the CLIPPER FORTRAN compiler is processed using the *as*(1) command resulting in a **.o** file (see *as*(1)).

.o Relocatable object file. The object file name is simply passed to the link edit pass.

Options

Before the description of each option and enclosed in parentheses, a restriction may be placed on the use of the option. The option is only to be used when that restriction applies.

-c Suppress the link edit pass of the compilation and force an object file to be produced even if only one program is compiled.

-C Turn on run-time checking of subranges and array bounds. The code will be much slower under this option.

-D *name* (**.F** file only) Define *name* to the preprocessor with the value **1**. This is equivalent to putting

#define *name* **1**

at the top of the source file.

-D*name*=*string*
(**.F** file only) Define *name* to the preprocessor with the value *string*. This is equivalent to putting

#define *name string*

at the top of the source file.

-E *xxx* Pass the string *xxx* to EFL as an option when preprocessing **.e** files into **.f** files.

-F Do not produce assembly, object, or executable files. Produce only FORTRAN source files. For each **.F** source language file, preprocess the file with the C preprocessor and leave the preprocessor output on a file whose name ends in **.f**. Similarly, preprocess each **.e** file with the EFL preprocessor and each **.r** file with the RATFOR preprocessor.

-g Cause the compiler to generate additional information needed for the use of source language debuggers like *sdb*(1). A frame pointer is generated to facilitate stack backtracing.

-ga A frame pointer is generated, but **-ga** does not produce the extra debugging information that is generated when **-g** is specified.

-I *dir* Add the directory *dir* to the list of directories searched for **include** file names that are not absolute (do not start with /). Multiple **-I** options can be specified, and each directory will be searched, in the order encountered, before a standard list of directories is searched.

-i2 Make the type INTEGER be INTEGER*2 and the type LOGICAL be LOGICAL*2. By default, INTEGER is the type INTEGER*4 and LOGICAL is the type LOGICAL*4.

-m Process RATFOR (**.r**) and EFL (**.e**) files with *m4*(1) before running the appropriate preprocessor.

-o *file-name*
Place the executable binary output from the link edit pass into the file named *file-name*. If this option is not specified the executable file will be named **a.out**. This option is ignored if **-c**, **-S**, or **-F** is present.

-onetrip Execute at least one iteration of every DO loop. The default case assumes that if the lower bound exceeds the upper bound, no iterations of the DO loop are to be performed (as specified by the ANSI FORTRAN-77 standard). The resolution of this case was unspecified under the ANSI FORTRAN-66 standard and some important implementations (especially IBM) chose to always execute the loop at least once. The use of this option makes the compiler incompatible with the ANSI FORTRAN-77 standard, but it may be necessary for compatible processing when certain old FORTRAN-66 programs are involved.

-1 (*one*) Equivalent to **-onetrip**.

-O The **-O** option activates Green Hills optimizers that are safe to use on all programs, except for the loop optimizer.

-OM This option is equivalent to **-O** except that it also allows the optimizer to assume that memory locations do not change except by explicit stores. That is, the optimizer is guaranteed that no memory locations are I/O device registers that can be changed by external hardware and no memory locations are shared with other processes that can change them asynchronously with respect to the current process. This compile time option must be used with extreme caution (or not at all) in device drivers, operating systems, shared memory environments, and when interrupts (or CLIX signals) are present.

-OL Optimize the program to be as fast as possible even if the program must be bigger. In particular, most of the available resources are allocated to optimizations of the innermost loops. The **-OL** compile time option will perform optimizations that may make the program faster but larger. It is counter-productive to specify **-OL** on code that contains no loops or that is rarely executed as it will make the whole program larger but no faster. After experimenting with a program, it is possible to discover which modules benefit from **-OL** and which ones do not.

-OLM This option is equivalent to **-OL** and **-OM**.

-OML This option is equivalent to **-OLM**.

-p Arrange for the compiler to produce code that counts the number of times each routine is called. If link editing occurs, replace the standard libraries with libraries compiled for profiling. Also, if link editing, replace the standard startoff routine by one that automatically calls *monitor*(3C) at the start and arranges to write out a **mon.out** file at normal termination of execution of the object program. An execution profile can then be generated by using *prof*(1).

-R *str* Pass the string *str* to the RATFOR preprocessor as an option when translating **.r** files into **.f** files.

-S Compile the named source programs and leave the assembler-language output on corresponding files suffixed by **.s**. The assembler and link edit passes are suppressed.

-u Make the default data type for undeclared variables undefined. This is equivalent to putting

```
IMPLICIT UNDEFINED(A-Z)
```

at the top of the source file.

-U Do not convert uppercase letters in names to lowercase. By default, FORTRAN is not case sensitive and all externally visible FORTRAN names contain only lowercase letters. This option can be used to gain access to external names that contain uppercase. However, using this option makes the compiler incompatible with the ANSI FORTRAN-77 standard.

-v Print the program name and command line arguments as each pass is invoked.

-w Suppress warning diagnostics.

-W*c*,*arg1* [,*arg2* ...]
Pass the listed argument(s) *argi* to phase *c* where *c* is one of [**p01almerc**]. (See the section on compilation phases above.)

-X *n* Turn on compile-time option number *n*. The available compile-time options are listed below.

9 Disable the local (peephole) optimizer.

18 Do not allocate programmer-defined local variables to a register. This option suppresses optimizations that frustrate debuggers.

32 Display the names of files as they are opened. This is useful for determining why the compiler cannot find an include file.

37 Emit a warning when dead code is eliminated.

39 Do not move frequently used procedure and data addresses to registers.

50 Push arguments on the stack. The default is to pass the first two arguments in registers. This option is not recommended because it produces a calling sequence incompatible with the rest of the CLIX System.

54 Inform the optimizer that no memory locations can change value asynchronously with respect to the running program. **-O2** sets this compile-time option (see **-O2** above).

58 Do not put an underscore in front of the names of global variables and procedures. This option is not recommended because it produces symbols incompatible with the rest of the CLIX System.

62 (Default) The target processor is a CLIPPER microprocessor.

68 This makes characters unsigned as they are in some implementations of FORTRAN. The default is signed characters.

71 Use the single precision math library interface.

74 (Default) The target system is CLIX System V.

77 Turn off compile-time checking of FORMAT statements. Use this option if the run-time library supports FORMAT statement features that the FORTRAN compiler is not aware of.

79 Pad Hollerith constants on the right with blanks. The default is that only the first byte of the Hollerith is significant and the constant is zero padded on the left.

80 Disable the code hoisting optimization. This can speed up compilation in some cases.

82 Process lines starting with **x**, **X**, **d**, and **D**. The default is to treat them as comments. Used for enabling debugging statements.

87 Disable the optimization that deletes all code that stores into or modifies variables that are never read from.

89 Pack structures with no space between members even if doing so makes the elements inaccessible due to machine data alignment constraints.

105 Allow **#define** symbols to be redefined to the preprocessor.

151 Do not allow dollar signs in names. The default allows dollars signs for VMS compatibility.

168 Do not move invariant floating-point expressions out of loops.

175 (Default) For System V compatibility, name the main program **MAIN___** . If this option is not specified or turned off (with **-Z175**), the name for the main program is **MAIN_**.

176 Always convert computations involving floating-point values to **DOUBLE PRECISION**. By default, the compiler tries to shorten computations to **REAL** if the result would be the same.

190 Assume half-word objects are not aligned.

191 Assume word objects are not aligned.

192 Assume single precision objects are not aligned.

193 Assume double precision objects are not aligned.

194 Assume word objects are aligned only to half-word boundaries.

195 Assume single precision objects are aligned only to half-word boundaries.

196 Assume double precision objects are aligned only to half-word boundaries.

197 Assume double precision objects are aligned only to word boundaries.

-Y [p012aclSILU], *dirname*

Use *dirname* to locate the phase(s) or directory(ies) specified by the key letter(s). The key letters **[p012acl]** represent the phases described above. The additional key letters have the following meanings:

S The directory containing the startup routines.

I The default directory searched for the **#include** preprocessor directives.

L The first default library directory searched (see *ld*(1)).

U The second default library directory searched (see *ld*(1)).

If the location of a phase is being specified, the new path name for the phase will be *dirname/phasename*. The exact name used for *phasename* depends on the compiler driver used and the phase involved. See **FILES** below. If more than one **-Y** option is applied to a phase or directory, the last specification is used.

-Z *n*	Turn off option number *n*. This is the reverse of the -X option. This option is useful if a version of the compiler has an option that is turned on by default, and the user wants to turn it off.
-#	Print the program name and command line arguments as each pass is invoked.
-##	Verbose like -#, only more so.
-###	Print the program name and command line arguments for each pass, but do not invoke the pass.

FILES

file.f	FORTRAN source input file
file.F	FORTRAN source input file, C preprocessor used
file.e	EFL source input file
file.r	RATFOR source input file
file.c	C source input file
file.s	assembler source input file; generated or input
file.o	object file; generated or input
a.out	default linked output
/tmp/F77*	temporary
/usr/tmp/F77*	temporary
/bin/cc	C compiler
/usr/bin/RATFOR	RATFOR preprocessor
/usr/bin/efl	EFL preprocessor
/usr/bin/m4	m4 macro-processor
/bin/as	assembler, *as*(1)
/bin/ld	link editor, *ld*(1)
/lib/crt[1n].o	run-time start-off
/lib/mcrt[1n].o	profiling start-off
/lib/libF77.a	standard FORTRAN library
/lib/libm.a	standard math library
/lib/libc.a	standard C library
/usr/lib/libbsd.a	BSD support library (referenced by /lib/libf.a)
/usr/lib/libp/lib*.a	profiled versions of libraries
/lib/libf.a	Green Hills FORTRAN library

SEE ALSO

adb(1), cc(1), as(1), ld(1), sdb(1), ratfor(1), efl(1).
prof(1), m4(1), monitor(3C) in the *UNIX System V Programmer's Reference Manual*.
The "Release Notes" appendix of the *CLIPPER FORTRAN Reference Manual*.

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the various preprocessor, C compiler, assembler, or link editor passes.

NAME

find - find files

SYNOPSIS

find *path-name-list expression*

DESCRIPTION

find recursively descends the directory hierarchy for each path name in the *path-name-list* (that is, one or more path names) seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer, where +*n* means more than *n*, -*n* means less than *n* and *n* means exactly *n*. Valid expressions are as follows:

-name *file*	True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for **[**, **?**, and *****).
[**-perm**] *-onum*	True if the file permission flags are identical to the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, only the bits set in *onum* are compared with the file permission flags and the expression evaluates true if they match.
-type *c*	True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **l**, **p**, or **f** for block special file, character special file, directory, symbolic link, fifo (also known as named pipe), or plain file respectively.
-links *n*	True if the file has *n* links.
-user *uname*	True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is interpreted as a user ID.
-group *gname*	True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is interpreted as a group ID.
-size *n*[**c**]	True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.
-atime *n*	True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by *find*.
-mtime *n*	True if the file has been modified in *n* days.
-ctime *n*	True if the file has been changed in *n* days.
-exec *cmd*	True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument **{}** is replaced by the current path name.
-ok *cmd*	Resembles **-exec** except that the generated command line is printed with a question mark first and is executed only if

	the user responds by typing a "y".
-print	Always true; prints the current path name.
-cpio *device*	Always true; write the current file on *device* in *cpio*(1) format (5120-byte records).
-newer *file*	True if the current file was modified more recently than the argument *file*.
-depth	Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*(1) to transfer files contained in directories without write permission.
-mount	Always true; restricts the search to the file system containing the directory specified. (or if no directory was specified, the current directory.)
-local	True if the file physically resides on the local system.
(*expression*)	True if the parenthesized expression is true. (Parentheses are special to the shell and must be escaped.)

The primaries may be combined using the following operators (in the order of decreasing precedence):

1) The negation of a primary (**!** is the unary *not* operator).

2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

3) Alternation of primaries (**-o** is the *or* operator).

EXAMPLES

To remove all files named **a.out** or ***.o** that have not been accessed for a week:

```
find / \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1), cpio(1), test(1), fs(4).
stat(2), umask(2) in the *UNIX System V Programmer's Reference Manual.*
sh(1) in the *UNIX System V User's Reference Manual.*

BUGS

find / -depth always fails with the message:

```
find: stat failed: No such file or directory.
```

NAME

fmu - network file management utility

SYNOPSIS

fmu [-cefaiqrsvx] [*host*.*user*[.[*password*]] [*command*]]

DESCRIPTION

fmu is a network file transfer and remote command utility. The *host* may be either a node name or an Ethernet address. Once the remote connection has been established, the privileges and working directory are the same as if a *login*(1) for the specified *user* occurred on the remote system.

For security, *fmu* requires a login, specified by *user*, on the remote system. If the login on the remote system has a password, *password* must be supplied. If a **.** followed by a <RETURN> or white space is specified after *user*, *fmu* will prompt for a *password* (with echoing disabled). Otherwise, the word immediately following the **.** is used as the password.

fmu has an interactive and a noninteractive mode. To use noninteractive mode, *host*, *user*, and *command* must be specified on the command line. Pipes can also be used on the command line.

Interactive mode is entered when *command* is not specified. After a user enters interactive mode, an **FMU>** prompt appears.

The following options are available:

-a Inhibit automatic compression when transferring files. By default, if *fmu* detects a slow transfer rate, it will automatically compress before being sent.

-c Force the output file to be created contiguously. This feature is valid only if supported by the receiving system.

-e Echo all commands before executing them. This feature is useful to view commands if **stdin** has been redirected from a file.

-f Force the output file to be created in fixed-length, 512-byte records. This feature is valid only if supported by the receiving system.

-i Designate files being transferred as Interactive Graphics Design System (IGDS) files. This option is equal to specifying both the **-c** and **-f** options. This feature is valid only if supported by the receiving system.

-p Execute the login profiles (**/etc/profile** and the user's **.profile**) on the remote system. If either profile requires input, *fmu* will not succeed in connecting to the remote system.

-q Suppress the **FMU>** prompt. This is useful if **stdin** is redirected from a file.

-r Print the *fmu* release date (version number).

-s	Turn on software checksumming when transferring files.
-v	Turn on verbose mode. When transferring files, *fmu* will print statistics of the files being transferred.
-x	Turn on compression when transferring files.

command specifies a function to be performed. Only as many characters as needed to uniquely identify a *command* need to be specified.

The following *command*s are available:

receive *in* [*out*]	Receive a file or multiple files from the remote *host*. The *in* parameter represents the file(s) on the remote *host* to be received and *out* is the name of the output file or directory on the local machine. If *in* consists of multiple files, *out* must be a directory. If *out* is not specified, the current directory is used. If this command is specified on the command line, received data may be sent to **stdout** if *out* is -.
send *in* [*out*]	Send a file or multiple files to the remote *host*. The *in* parameter represents the file(s) on the local machine to be sent and *out* is the name of the output file (or directory) on the remote *host*. If *in* consists of multiple files, *out* must be a directory. If *out* is not specified, the current directory is used. If this command is specified on the command line, data to be sent may be received from **stdin** if *in* is -.
cat *file* ...	Display the remote *file*s to **stdout** on the local machine.
connect *nodespec*	Terminate the present connection (if any) and establish a connection with the remote system specified by *nodespec*. The *nodespec* syntax has the same form as *host*.*user*[.[*password*]] specified on the command line.
ls [*dir*]	List the contents of the directory *dir* on the remote *host*. The argument *dir* is passed to the directory listing program on the remote *host*, which checks for proper syntax. If *dir* is not provided, the current directory is used.
rm *file* ...	Remove the specified *file*s from the remote *host*.
cd [*dir*]	Change to the directory *dir* on the local machine. If *dir* is not provided, the environment variable HOME is used.
rcd [*dir*]	Change to the directory *dir* on the remote *host*. If *dir* is not provided, the environment variable HOME on the remote *host* is used.
command *string*	Execute *string* on the remote *host*. All **stdout** written by the remote command will be written to **stdout** on the local machine.
!*command*	Execute *command* on the local machine.

help [*arg*]	Print a one-line summary of *arg*. If *arg* is not provided, all available commands are listed.
exit	Close the current connection (if any) and exit.
set *option*	Set the specified *option*. Valid *options* are as follows:

[**no**]**checksum**	Function the same as the **-s** option.
[**no**]**compress**	Function the same as the **-x** option.
[**no**]**contiguous**	Function the same as the **-c** option.
[**no**]**echo**	Function the same as the **-e** option.
[**no**]**fixed**	Function the same as the **-f** option.
[**no**]**igds**	Function the same as the **-i** option.
[**no**]**inhibit**	Function the same as the **-a** option.
[**no**]**quiet**	Function the same as the **-q** option.
[**no**]**verbose**	Function the same as the **-v** option.

type *file* ...	Synonym for **cat**.
directory [*dir*]	Synonym for **ls**.
delete *file* ...	Synonym for **rm**.
chdir [*dir*]	Synonym for **cd**.
rchdir [*dir*]	Synonym for **rcd**.

Wildcards can be used when specifying files. However, for output files, no partial wildcard specifications are accepted. For example, ***** is a valid output file specification, but ***.txt** is not. Wildcards entered on the command line must be quoted to prevent the shell from expanding them.

While in interactive mode command-line recall and editing features are available. The key definitions are as follows:

<RETURN>	Recall and execute the most recent command.
<CONTROL>-A	Go to the beginning of the line.
<CONTROL>-E	Go to the end of the line.
<CONTROL>-D	Delete the character the cursor is on. If no command is being edited, this key sequence will terminate the session.
<CONTROL>-P **<UP-ARROW>**	Recall the previous command.
<CONTROL>-K	Delete all characters to the right of the cursor.
<CONTROL>-N **<DOWN-ARROW>**	Recall the next command.
<CONTROL>-B **<LEFT-ARROW>**	Move the cursor to the left one position.
<CONTROL>-F **<RIGHT-ARROW>**	Move the cursor to the right one position.

<DELETE>	Delete the character to the left of the cursor.
<KILL>	Delete the entire line.
<EOF> **<RETURN>** **<LINE FEED>**	Designate the end of a command. *fmu* then executes the line contents.

EXAMPLES

The following command sends all files beginning with **foo** and ending with **c** to **/tmp** on the remote host **abc**:

```
fmu abc.guest send 'foo*.c' /tmp
```

foo*.c must be quoted because * is a shell special character.

The following command sends a *cpio*(1) backup to the file **backupfile** on the remote host **backup** using **stdin**:

```
find src -print | cpio -ov | fmu backup.guest send - backupfile
```

To restore the previous example, use the following command:

```
fmu backup.guest rec backupfile - | cpio -iv
```

The following example is an interactive session with *fmu* to execute the *who*(1) command on the remote *host*:

```
$ fmu 08-00-36-23-08-00.remote.guest
FMU> com who
root   console Dec 7 17:12
FMU> exit
$
```

SEE ALSO

rpipe(1).
fmus(1M) in the *CLIX System Administrator's Reference Manual.*
cat(1), ls(1), rm(1), sh(1) in the *UNIX System V User's Reference Manual.*

CAVEATS

If the remote *command* requires input, *fmu* will hang.

fmu does not handle the binary output of a remote command (such as *tar*(1) or *cpio*(1)).

NAME

format - floppy disk formatting utility

SYNOPSIS

/etc/format [**-wl**]

DESCRIPTION

format formats the floppy disk in the floppy drive. This operation prepares the floppy disk for subsequent writes by any utility.

The format operation consists of writing ID fields, gaps, and address marks for each block on the floppy disk. This servo information is then used to identify tracks and sectors during *read*(2) and *write*(2) operations.

The high density format provides 1.2M on the floppy disk (1.44M on a 3.5-in disk). This type of format consists of 80 tracks with fifteen (eighteen) 512-byte sectors on each side of the floppy disk. By default, *format* will use high density.

The low density format provides 360K on the floppy disk (720K on a 3.5-in disk). This type of format consists of 40 (80) tracks with nine 512-byte sectors on each side of the floppy disk.

The following options are supported:

-l	Indicate low density. (The default is high density.)
-w	Disable the warning and prompt.

FILES

/dev/rdsk/f1	default floppy device

SEE ALSO

fl(7S) in the *CLIX System Administrator's Reference Manual.*

WARNINGS

All data on the floppy disk will be overwritten during a format operation. By default, a warning and prompt are printed before the floppy is formatted.

NAME

ftp - ARPANET file transfer program

SYNOPSIS

ftp [**-v**] [**-d**[*value*]] [**-i**] [**-n**] [**-g**] [**-r**] [*host*[*port*]]

DESCRIPTION

ftp is the user interface to the ARPANET standard File Transfer Protocol (FTP). The program allows a user to transfer files to and from a remote or local network site.

The client host that *ftp* will communicate with can be specified on the command line. If the host is specified, *ftp* immediately attempts to establish a connection to an FTP server on that host. Otherwise, *ftp* enters its command interpreter and awaits instructions from the user. When *ftp* is awaiting commands from the user, the **ftp>** prompt is displayed.

If a host name is specified on the command line, an optional port number can be specified. In this case, *ftp* will attempt to connect an FTP server at that port.

Options

Options may be specified at the command line or to the command interpreter. The following options are available:

-v Show all responses from the remote server and report data transfer statistics.

-n Do not attempt auto-login on initial connection. If auto-login is enabled, *ftp* checks the **.netrc** file (see below) in the user's home directory for an entry describing parameters to be used in logging in to the remote machine. If no entry exists, *ftp* prompts for a login name to be used on the remote machine. If no login name is given, a login is attempted with the user's local login name. If the user name exists and has a password, *ftp* will prompt for a password and an account (see the **account** command).

-i Turn off interactive prompting during multiple file transfers.

-d [*value*] Enable debugging. If an integer *value* is specified, *value* becomes the debugging level.

-g Disable file name globbing. (Do not expand wildcard characters (see the **glob** command).)

-r Display the *ftp* version number.

The .netrc file

The **.netrc** file contains login and initialization information used by the auto-login process. It resides in the user's local home directory. The following tokens are recognized; they may be separated by spaces, tabs, or newlines:

machine *name*
: Identify a remote machine name. The auto-login process searches the **.netrc** file for a **machine** token followed by a *name* that matches the *host* given on the *ftp* command line or as an argument to the **open** command. Once a match is found, subsequent **.netrc** tokens are processed until the end of the file is reached or another **machine** token is encountered.

login *name*
: Identify a user name on the remote machine. If this token is present, the auto-login process will initiate a login using *name*.

password *string*
: Supply a password. If this token is present, the auto-login process will supply *string* if the remote server requires a password as part of the login process. Note that if this token is present in the **.netrc** file, *ftp* will abort the auto-login process if the **.netrc** file is readable by anyone other than the user.

account *string*
: Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password. The auto-login process will initiate an ACCT command if it does not.

macdef *name*
: Define a macro. This token functions as the *ftp* **macdef** command does. A macro called *name* is defined; its contents begin with the next **.netrc** line and continue until a null line (consecutive newline characters) is encountered. If a macro named **init** is defined, it is automatically executed as the last step in the auto-login process.

Commands

ftp recognizes the following commands:

! [*command* [*args*]]
: Invoke an interactive shell on the local machine. If there are arguments, the first is interpreted as a command to execute; the remaining arguments are the command's arguments.

$ *macro-name* [*args*]
: Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro unglobbed.

account [*passwd*]
: Supply a supplemental password required by a remote system to access resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a nonechoing input mode.

append *local-file* [*remote-file*]
: Append a local file to a file on the remote machine. If *remote-file* is unspecified, the local file name is used to name the remote file after the local file name is altered by any **ntrans** or **nmap** setting. File transfer uses the current settings for **type**, **form**, **mode**, and **struct**.

ascii Set the file transfer **type** to network ASCII. This is the default type.

bell Set the bell to be sounded after each file transfer command is completed.

binary Set the file transfer **type** to support binary image transfer.

bye Terminate the FTP session with the remote server and exit *ftp*. An end of file will also terminate the session and exit.

case Toggle remote computer file name case mapping during **mget** commands. When **case** is on (default is off), remote computer file names with all letters in uppercase are written in the local directory with the letters mapped to lowercase.

cd *remote-directory*
: Change the working directory on the remote machine to *remote-directory*.

cdup Change the remote machine working directory to the parent of the current remote machine working directory.

close Terminate the FTP session with the remote server and return to the command interpreter. Any defined macros are erased.

cr Toggle carriage-return stripping during **ascii**-type file retrieval. Records are denoted by a carriage-return/linefeed sequence during **ascii**-type file transfer. When **cr** is on (the default), carriage returns are stripped from this sequence to conform to the CLIX single linefeed record delimiter. Records on non-CLIX remote systems may contain single linefeeds; when an **ascii**-type transfer is made, these linefeeds may be distinguished from a record delimiter only when **cr** is off.

delete *remote-file*
: Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
: Set or toggle the debugging mode. If an optional *debug-value* is specified, this value sets the debugging level. If *debug-value* is not specified, debugging mode is toggled. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string **—>**. If *debug-value* is set to 1, a user's password will not be displayed.

dir [*remote-directory*] [*local-file*]
: Print a listing of the contents of the *remote-directory* and, optionally, place the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified or *local-file* is -, output comes to the terminal.

disconnect
: A synonym for **close**.

form *form*
: Set the file transfer **form** to *form*. The default format is **file**.

get *remote-file* [*local-file*]
: Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the name it has on the remote machine. However, the current **case**, **ntrans**, and **nmap** settings can alter this name. The current settings for **type**, **form**, **mode**, and **struct** are used while the file is being transferred.

glob
: Toggle file name expansion for **mdelete**, **mget** and **mput**. If globbing is turned off with **glob**, *ftp* interprets file name characters literally and not as wildcard characters. Globbing for **mput** is performed as it is in *sh*(1). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name will likely differ from expansion of an ordinary file name: the exact result depends on the foreign operating system and *ftp* server and can be previewed by executing **mls** *remote-files* -. **mget** and **mput** are not meant to transfer entire directory subtrees of files. This can be done by transferring a *tar*(1) archive of the subtree (in binary mode).

hash
: Toggle hash-sign (#) printing for each data block transferred. The size of a data block is 1024 bytes.

help [*command*]
: Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

lcd [*directory*]
: Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
: Print an abbreviated listing of a directory's contents on the remote machine. If *remote-directory* is unspecified, the current working directory is used. If no local file is specified or if *local-file* is -, the output is sent to the terminal.

macdef *macro-name*
: Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed. The macro processor interprets **$** and \ as special characters. A **$** followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A **$** followed by an **i** signals the macro processor that the executing macro is to be looped. On the first pass, **$i** is replaced by the first argument on the macro invocation command line; on the second pass it is replaced by the second argument; and so on. A \ followed by any character is replaced by that character. Use the \ to prevent special treatment of the **$**.

mdelete [*remote-files*]
: Delete the *remote-files* on the remote machine.

mdir *remote-files local-file*
: Resembles **dir** except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

mget *remote-files*
: Expand the *remote-files* on the remote machine and execute **get** for each file name produced. See **glob** for details on the file name expansion. Resulting file names will then be processed according to **case**, **ntrans**, and **nmap** settings. Files are transferred into the local working directory, which can be changed with **lcd** *directory*; new local directories can be created with **!** **mkdir** *directory*.

mkdir *directory-name*
: Make a directory on the remote machine.

mls *remote-files local-file*
: Resembles **ls**, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mls** output.

mode [*mode-name*]
: Set the file transfer **mode** to *mode-name*. The default mode is **stream** mode.

mput *local-files*
: Expand wild cards in the list of local files given as arguments and execute **put** for each file in the resulting list. See **glob** for details of file name expansion. Resulting file names will then be processed according to **ntrans** and **nmap** settings.

nmap [*inpattern outpattern*]
: Set or unset the file name mapping mechanism. If arguments are not specified, the file name mapping mechanism is unset. If arguments are specified, remote file names are mapped during **mput** commands and **put** commands issued without a specified remote target file name. If arguments are specified, local file names are mapped during **mget** commands and **get** commands issued without a specified local target file name. This command is useful when connecting to a non-CLIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *Inpattern* is a template for incoming file names (which may have been processed according to the **ntrans** and **case** settings). Variable templating is accomplished by including the sequences **$1**, **$2**, ... , **$9** in *inpattern*. A \ prevents this special treatment of the **$** character. All other characters are treated literally and determine the **nmap** *inpattern* variable values. For example, given *inpattern* **$1.$2** and the remote file name **mydata.data**, **$1** would have the value **mydata**, and **$2** would have the value **data**. The *outpattern* determines the resulting mapped file name. The sequences **$1**, **$2**, ... , **$9** are replaced by any value resulting from the *inpattern* template. The sequence **$0** is replace by the original file name. Additionally, the sequence [*seq1*,*seq2*] is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*. For example, the command **nmap $1.$2.$3 [$1,$2].[$2,file]** would yield the output file name **myfile.data** for input file names **myfile.data** and

myfile.data.old, **myfile.file** for the input file name **myfile**, and **myfile.myfile** for the input file name **.myfile**. Spaces may be included in *outpattern* as in the example **nmap $1 | sed "s/ *$//" > $1**. A **\\** prevents special treatment of the **$**, **[**, **]**, and **,** characters.

ntrans [*inchars* [*outchars*]]
Set or unset the file name character translation mechanism. If arguments are not specified, the file name character translation mechanism is unset. If arguments are specified, characters in remote file names are translated during **mput** commands and **put** commands issued without a specified remote target file name. If arguments are specified, characters in local file names are translated during **mget** commands and **get** commands issued without a specified local target file name. This command is useful when connecting to a non-CLIX remote computer with different file naming conventions or practices. Characters in a file name matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

open *host* [*port*]
Establish a connection to the specified *host* FTP server. An optional *port* number may be supplied. If a port number is specified, *ftp* will attempt to contact an FTP server at that *port*. If the *auto-login* option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see below).

prompt Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default is on), any **mget** or **mput** will transfer all files, and any **mdelete** will delete all files.

proxy *ftp-command*
Execute an *ftp* command on a secondary control connection. This command allows simultaneous connection to two remote FTP servers for transferring files between the two servers. The first **proxy** command should be an **open** to establish the secondary control connection. Enter the command **proxy ?** to see other *ftp* commands that are executable on the secondary connection. The following commands behave differently when prefixed by **proxy**: **open** will not define new macros during the auto-login process; **close** will not erase existing macro definitions; **get** and **mget** transfer files from the host on the primary

control connection to the host on the secondary control connection; and **put**, **mput**, and **append** transfer files from the host on the secondary control connection to the host on the primary control connection. Third-party file transfers depend on the server on the secondary control connection supporting the FTP protocol PASV command.

put *local-file* [*remote-file*]
: Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any **ntrans** or **nmap** settings in naming the remote file. File transfer uses the current settings for **type**, **form**, **mode**, and **struct**.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for **bye**.

quote *arg1 arg2 ...*
: The specified arguments are sent, verbatim, to the remote FTP server.

recv *remote-file* [*local-file*]
: A synonym for **get**.

remotehelp [*command-name*]
: Request help from the remote FTP server. If a *command-name* is specified, it is supplied to the server.

rename [*from*] [*to*]
: Rename the file *from* on the remote machine to the file *to*.

reset Clear the reply queue. This command resynchronizes command/reply sequencing with the remote FTP server. Resynchronization may be necessary following a violation of the FTP protocol by the remote server.

rmdir *directory-name*
: Delete a directory on the remote machine.

runique Toggle storing of files on the local system with unique file names. If a file already exists with a name equal to the target local file name for a **get** or **mget** command, a **.1** is appended to the name. If the resulting name matches another existing file, a **.2** is appended to the original name. If this process continues up to **.99**, an error message is printed and the transfer does not occur. The generated unique file name will be reported. Note that **runique** will not affect local files generated from a shell command (see below). The default value is off.

send *local-file* [*remote-file*]
: A synonym for **put**.

sendport Toggle the use of PORT commands. By default, *ftp* will attempt to use a PORT command when establishing a connection for each data transfer. PORT commands can prevent delays during multiple file transfers. If the PORT command fails, *ftp* will use the default data port. When PORT commands are disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations that ignore PORT commands but incorrectly indicate that they have been accepted.

status Show the current status of *ftp*.

struct [*struct-name*]
Set the file transfer **struct** to *struct-name*. By default, **stream** structure is used.

sunique Toggle storing of files on remote machine under unique file names. The remote FTP server must support the FTP protocol STOU command for successful completion. The remote server will report a unique name. The default value is off.

tenex Set the file transfer type to that needed to communicate with TENEX machines.

trace Toggle packet tracing. This mode is not currently implemented.

type [*type-name*]
Set the file transfer **type** to *type-name*. If no type is specified, the current type is printed. The default type is network **ascii**.

user *user-name* [*password*] [*account*]
Identify the user to the remote FTP server if the **-n** option is not used. If the password is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an account field is not specified and the FTP server requires it, the user will be prompted for it. If an account field is specified, an ACCT command will be relayed to the remote server after the login sequence is complete if the remote server did not require it for logging in. Unless *ftp* is invoked with auto-login disabled, this process occurs automatically when the FTP server is initially connected to. If an invalid user name or password was given in the auto-login process or in a previous **user** command, the **user** command should be used to specify a valid *user-name* (and *password*, if necessary).

verbose Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]
: A synonym for **help**.

Command arguments that have embedded spaces may be quoted with quotation (" ") marks.

Aborting File Transfer

To abort a file transfer, press the terminal interrupt key (usually <CONTROL>-C). Sending transfers will be immediately halted. Receiving transfers will be halted by sending an FTP protocol ABOR command to the remote server and discarding any further data received. The speed at which this is accomplished depends on the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an **ftp>** prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when *ftp* has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above or from the remote server behaving unexpectedly, including violation of the FTP protocol. If the delay results from unexpected remote server behavior, the local *ftp* program must be killed manually.

File Naming Conventions

Files specified as arguments to *ftp* commands are processed according to the following rules.

1) If the file name - is specified, **stdin** (for reading) or **stdout** (for writing) is used.

2) If the first character of the file name is |, the remainder of the argument is interpreted as a shell command. *ftp* then forks a shell using *popen*(3C) with the argument supplied. It then reads (writes) from **stdout (stdin)**. If the shell command includes spaces, the argument must be quoted as in "**| ls -lt**". A useful example of this mechanism is "**dir * |pg**".

3) Failing the above checks, if globbing is enabled, local file names are expanded according to the rules used in the *sh*(1) (see the **glob** command). If the *ftp* command expects a single local file (such as **put**), only the first file name generated by the globbing operation is used.

4) For **mget** and **get** commands with unspecified local file names, the local file name is the remote file name, which may be altered by a **case**, **ntrans**, or **nmap** setting. The remote server may then alter the resulting file name if **runique** is on.

5) For **mput** and **put** commands with unspecified remote file names, the remote file name is the local file name, which may be altered by a

ntrans or **nmap** setting. The resulting file name may then be altered by the remote server if **sunique** is on.

File Transfer Parameters

The FTP specification specifies many parameters that may affect a file transfer. The **type** may be **ascii, image** (binary), **ebcdic**, or **local byte size** (for PDP-10s and PDP-20s mostly). *ftp* supports the **ascii** and **image** types of file transfer plus **local byte size 8** for **tenex** mode transfers. *ftp* treats **image** and **tenex 8** file types the same when transferred.

ftp supports only the default values for the remaining file transfer parameters: **mode, form**, and **struct.**

SEE ALSO

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

WARNINGS

Correct command execution depends on the remote server behaving properly.

NAME

hostname - set or print name of current host system

SYNOPSIS

hostname [*nameofhost*]

DESCRIPTION

The *hostname* command prints the name of the current host, as given before the "login" prompt, and defaults to the value set on the "Operating System Parameters" page in the Startup Utility. The super-user may change the host name by giving an argument.

SEE ALSO

gethostname(2B), sethostname(2B).

CAVEATS

Permanent host name changes can only be made from the Startup Utility.

NAME

jbconfig - report the configuration of the jukeboxes

SYNOPSIS

jbconfig

DESCRIPTION

jbconfig lists the current jukebox configuration including all jukeboxes, drives, and volumes.

SEE ALSO

JBCFG(4).
jbinventory(1M) in the *CLIX System Administrator's Reference Manual.*

CAVEATS

If numerous volumes are in the database, the report can be lengthy.

NAME

ident - identify files

SYNOPSIS

ident [**-q**] [*file* ...]

DESCRIPTION

ident searches the named files or, if no file name appears, the standard input for all occurrences of the pattern **$***keyword*:...**$**, where *keyword* is one of the following:

Author
Date
Header
Id
Locker
Log
Revision
RCSfile
Source
State

These patterns are normally inserted automatically by the RCS command *co*(1), but can also be inserted manually. The option **-q** suppresses the warning given if no patterns are in a file.

ident works on text files, object files, and dumps. For example, if the C program in file **f.c** contains

```
char rcsid[] = "$Header: Header information $";
```

and **f.c** is compiled into **f.o**, then the command

```
ident f.c f.o
```

will print

```
f.c:
        $Header: Header information $
f.o:
        $Header: Header information $
```

SEE ALSO

ci(1), co(1), rcs(1), rcsclean(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(4).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

IDENTIFICATION

Author: Walter F. Tichy,
Purdue University, West Lafayette, IN 47907
Copyright © 1982 by Walter F. Tichy.

NAME

kbmap - change the keyboard layout

SYNOPSIS

kbmap *mapfile*

DESCRIPTION

kbmap changes keyboard layouts to reflect the standard layouts used in different cultures. Only the letter, number, and punctuation keys on the main section of the keyboard are affected. The keypad and function keys are not changed.

EXAMPLES

To change the keyboard layout to a standard French layout, use the following command:

```
kbmap /usr/lib/kbmap/French
```

FILES

/usr/lib/kbmap/*	files for each keyboard type

SEE ALSO

kbmap(4).

NAME

kermit - kermit file transfer

SYNOPSIS

kermit [*option* ...] [*file* ...]

DESCRIPTION

kermit is a file transfer program that allows files to be moved among machines of different operating systems and architectures. This manual page describes version 4C of the program.

Arguments are optional. If *kermit* is executed without arguments, it enters command mode. Otherwise, *kermit* reads the arguments from the command line and interprets them.

The following notation is used in command descriptions:

fn	A CLIX file specification, possibly containing either of the wildcard characters * or ?. (* matches all character strings; ? matches any single character.)
fn1	A CLIX file specification that may not contain * or ?.
rfn	A remote file specification in the remote system's syntax, which may denote a single file or a group of files.
rfn1	A remote file specification should denote only a single file.
n	A decimal number between 0 and 94.
c	A decimal number between 0 and 127 representing the value of an ASCII character.
cc	A decimal number between 0 and 31, or else exactly 127, representing the value of an ASCII control character.
[ ]	Any field in square braces is optional.
{*x,y,z*}	Alternatives are listed in braces.

kermit command line options may specify either actions or settings. If *kermit* is invoked with a command line that specifies no actions, it will issue a prompt and begin interactive dialog. Action options specify either protocol transactions or terminal connection.

Command Line Options

-s *fn* Send the specified file or files. If *fn* contains wildcard (meta) characters, the shell expands it to a list. If *fn* is -, *kermit* sends from standard input, which must come from a file:

```
kermit -s - < foo.bar
```

or a parallel process:

```
ls -l | kermit -s -
```

This mechanism cannot be used to send from the terminal keyboard. To send a file whose name is -, precede it with a path name, as in

```
kermit -s ./-
```

-r Receive a file or files. Wait passively for files to arrive.

-k Receive (passively) a file or files, sending them to standard output. This option can be used in several ways:

```
kermit -k
```

Displays the incoming files on the screen; to be used only in local mode (see below).

```
kermit -k > fn1
```

Sends the incoming file or files to the named file, *fn1*. If more than one file arrives, all are concatenated together into the single file *fn1*.

```
kermit -k | command
```

Pipes the incoming data (single or multiple files) to the indicated command, as in

```
kermit -k | sort > sorted.stuff
```

-a *fn1* If a file transfer option is specified, an alternate name for a single file may be specified with the **-a** option. For example,

```
kermit -s foo -a bar
```

sends the file **foo** telling the receiver that its name is **bar**. If more than one file arrives or is sent, only the first file is affected by the **-a** option:

```
kermit -ra baz
```

stores the first incoming file under the name **baz**.

-x Begin server operation. May be used in either local or remote mode.

kermit is local if it is running on a PC directly, or if it is running on a multiuser system and transferring files over an external communication line—not the job's controlling terminal or console. *kermit* is remote if it is running on a multiuser system and transferring files over its own controlling terminal's communication line connected to a PC.

If *kermit* is running on a PC, it is in local mode by default, with the "back port" designated for file transfer and terminal connection. If *kermit* is running on a multiuser (timesharing) system, it is in remote mode unless explicitly pointed at an external line for file transfer or terminal connection. The following command sets *kermit*'s mode:

-l *dev* Line — Specify a terminal line to use for file transfer and terminal connection, as in

```
kermit -l /dev/tty00
```

When an external line is used, some additional options may be used for successful communication with the remote system:

-b *n* Baud — Specify the baud rate for the line given in the -l option, as in

```
kermit -l /dev/tty00 -b 9600
```

This option should always be included with the -l option, since the speed of an external line is not always as expected.

-p *x* Parity — **e**, **o**, **m**, **s**, **n** (even, odd, mark, space, or none). If parity is other than none, the eighth-bit prefixing mechanism will be used for transferring eight-bit binary data if the opposite *kermit* agrees. The default parity is none.

-t Specifies half-duplex line turnaround with XON as the handshake character.

The following commands may be used only with a *kermit* that is local—either by default or because the -l option has been specified.

-g *rfn* Actively request a remote server to send the named file or files; *rfn* is a file specification in the remote host's syntax. If *rfn* contains any special shell characters like "*" these must be quoted, as in

```
kermit -g "x*.?"
```

-f Send a **finish** command to a remote server.

-c Establish a terminal connection over the specified or default communication line before any protocol transaction occurs. Return to the local system by typing the escape character (normally <CONTROL>-\) followed by the letter **c**.

-n Resembles **-c**, but after a protocol transaction occurs; **-c** and **-n** may both be used in the same command. The use of **-n** and **-c** is illustrated below.

On a timesharing system, the **-l** and **-b** options must also be included with the **-r**, **-k**, or **-s** options if the other *kermit* is on a remote system.

If *kermit* is in local mode, the screen (**stdout**) is continuously updated to show the progress of the file transfer. A dot is printed for every four data packets. Other packets are shown by type (i.e., **S** for Send-Init), **T** is printed when a timeout occurs, and **%** is printed for each retransmission. In addition, certain "interrupt" commands may be typed (to **stdin**) during file transfer:

<CONTROL>-F	Interrupt the current file and proceed to the next (if any).
<CONTROL>-B	Interrupt the entire batch of files and terminate the transaction.
<CONTROL>-R	Resend the current packet.
<CONTROL>-A	Display a status report for the current transaction.

These interrupt characters differ from the ones used in other *kermit* implementations to avoid conflict with CLIX shell interrupt characters. With System III and System V implementations of UNIX, interrupt commands must be preceded by the escape character (e.g., <CONTROL>-\).

Several other command line options are provided:

-i Specifies that files should be sent or received exactly "as is" with no conversions. This option is necessary for transmitting binary files. It may also be used to slightly boost efficiency in UNIX-to-UNIX text file transfers by eliminating CRLF/newline conversion.

-w Write-Protect — Avoid file name collisions for incoming files.

-q Quiet — Suppress screen update during file transfer. (For instance, to allow a file transfer to proceed in the background.)

-d Debug — Record debugging information in the file **debug.log** in the current directory. Use this option if the program is not working properly.

-h Help — Display a brief synopsis of the command line options.

The command line may contain no more than one protocol action option.

Interactive Operation

kermit's interactive command prompt is "C-Kermit>". In response to this prompt, any valid command may be entered. *kermit* executes the command and prompts for another command. The process continues until the user instructs the program to terminate.

Commands begin with a keyword, normally an English verb, like **send**. Trailing characters may be omitted from any keyword so long as sufficient characters are specified to distinguish it from any other keyword valid in that field. Certain commonly-used keywords (such as **send**, **receive**, and **connect**) have special nonunique abbreviations (**s** for **send**, **r** for **receive**, and **c** for **connect**).

Certain characters have special functions in interactive commands:

? A question mark typed at any point in a command, will produce a message explaining the action possible or expected at that point. Depending on the context, the message may be a brief phrase, a menu of keywords, or a list of files.

<ESC> (The Escape or Altmode key) — Request completion of the current keyword or file name, or insertion of a default value. The result will be a beep if the requested operation fails.

<DEL> (The Delete or Rubout key) — Delete the previous character from the command. A <BACK SPACE> (back space key or <CONTROL>-H) may also be used for this function.

<CONTROL>-W Erase the right-most word from the command line.

<CONTROL>-U	Erase the entire command.
<CONTROL>-R	Redisplay the current command.
<SPACE>	(Space bar) — Delimits fields (keywords, file names, numbers) within a command. **<TAB>** (horizontal tab) may also be used for this purpose.
<RETURN>	(Carriage return) — Enters the command for execution. **<LINE FEED>** (linefeed) or **<FF>** (formfeed) may also be used for this purpose.
\	Enter any of the above characters in the command, literally. To enter a backslash, type two backslashes (\\). A single backslash immediately preceding a carriage return allows the continuation of the command on the next line.

The editing characters (i.e., <DEL> and <CONTROL>-W) may be typed repeatedly to delete to the prompt. No action will be performed until the command is entered by typing carriage return, linefeed, or formfeed. If mistakes are made and an informative error message and a new prompt are received, it is advisable to use the **help** command.

Interactive *kermit* accepts commands from files and the keyboard. When interactive mode is entered, *kermit* looks for the file **.kermrc** first in the home directory and then in the current one and executes any commands it finds. These commands must be in interactive format, not CLIX command line format. A **take** command is also provided to use at any time during an interactive session. Command files may be nested to any reasonable depth.

Here is a brief list of *kermit* interactive commands:

!	Execute a CLIX shell command.
bye	Terminate and log out from a remote *kermit* server.
close	Close a log file.
connect	Establish a terminal connection to a remote system.
cwd	Change working directory.
dial	Dial a telephone number.
directory	Display a directory listing.
echo	Display arguments literally.
exit	Exit from the program, closing any open logs.
finish	Instruct a remote *kermit* server to exit, but not log out.
get	Get files from a remote *kermit* server.

help	Display a help message for a given command.
log	Open a log file — debugging, packet, session, transaction.
quit	Same as **exit**.
receive	Passively wait for files to arrive.
remote	Issue file management commands to a remote *kermit* server.
script	Execute a login script with a remote system.
send	Send files.
server	Begin server operation.
set	Set various parameters.
show	Display values of **set** parameters.
space	Display current disk space usage.
statistics	Display statistics about most recent transaction.
take	Execute commands from a file.

The **set** parameters are:

block-check	Level of packet error detection.
delay	How long to wait before sending first packet.
duplex	Specify which side echoes during **connect**.
escape-character	Character to prefix escape commands during **connect**.
file	Set various file parameters.
flow-control	Communication line full-duplex flow control.
handshake	Communication line half-duplex turnaround character.
line	Communication line device name.
modem-dialer	Type of modem-dialer on communication line.
parity	Communication line character parity.
prompt	Change the *kermit* program's prompt.
receive	Set various parameters for inbound packets.
send	Set various parameters for outbound packets.
speed	Communication line speed.

The **remote** commands are:

cwd	Change remote working directory.
delete	Delete remote files.

directory	Display a listing of remote file names.
help	Request help from a remote server.
host	Issue a command to the remote host in its own command language.
space	Display the current disk space usage on remote system.
type	Display a remote file on the screen.
who	Display who's logged in or information about a user.

FILES

$HOME/.kermrc	*kermit* initialization commands
./.kermrc	more *kermit* initialization commands

SEE ALSO

cu(1C), uucp(1C) in the *UNIX System V User's Reference Manual.*
Frank da Cruz and Bill Catchings, *Kermit User's Guide*, Columbia University, 6th Edition

DIAGNOSTICS

The diagnostics produced by *kermit* are self-explanatory.

NOTES

See recent issues of the Info-Kermit digest (on ARPANET or Usenet) for a list of bugs.

NAME

ksh, krsh - shell, the standard/restricted command programming language

SYNOPSIS

ksh [-aefhikmnoprstuvx] [-o *option* **] ... [-c** *string* **] [***arg* **...]**
krsh [-aefhikmnoprstuvx] [-o *option* **] ... [-c** *string* **] [***arg* **...]**

DESCRIPTION

ksh is a command programming language that executes commands read from a terminal or a file. *krsh* is a restricted version of the standard command interpreter *ksh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See **Invocation** below for the meaning of arguments to the shell.

Definitions

A *metacharacter* is one of the following characters:

; & ( ) | < > newline space tab

A *blank* is a tab or a space. An *identifier* is a sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for *aliases*, *functions*, and *named parameters*. A *word* is a sequence of *characters* separated by one or more nonquoted *metacharacters*.

Commands

A *simple-command* is a sequence of *blank* separated words that may be preceded by a parameter assignment list. (See **Environment** below.) The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a simple-command is its exit status if it terminates normally or (octal) 200+*status* if it terminates abnormally. (See *signal*(2) for a list of status values.)

A *pipeline* is a sequence of one or more *commands* separated by **|**. The standard output of every command except the last is connected by a *pipe*(2) to the standard input of the next command. Each command runs as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **||** and optionally terminated by **;**, **&**, or **|&**. Of these five symbols, **;**, **&**, and **|&** have equal precedence, which is lower than **&&** and **||**. The symbols **&&** and **||** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (the shell does *not* wait for that pipeline to finish). The symbol **|&** causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell. The parent shell can read to and write from the standard input and output of the spawned command using the **-p** option of the special commands **read** and **print** described later. Only one such command can be active at any given time. The symbol **&&** (**||**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (nonzero) value. An arbitrary number of newlines may appear in a *list* instead of semicolons to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *identifier* [**in** *word* ...] **do** *list* **done**

Each time a **for** command is executed, *identifier* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, the **for** command executes the **do** *list* once for each positional parameter that is set (see **Parameter Substitution** below). Execution ends when no more words are in the list.

select *identifier* [**in** *word* ...] **do** *list* **done**

A **select** command prints on standard error (file descriptor 2) the set of *words* each preceded by a number. If **in** *word* ... is omitted, the positional parameters are used instead (see **Parameter Substitution** below). The **PS3** prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed **words**, the value of the parameter *identifier* is set to the *word* corresponding to this number. If this line is empty, the selection list is printed again. Otherwise the value of the parameter *identifier* is set to **null**. The contents of the line read from standard input is saved in the parameter **REPLY**. The *list* is executed for each selection until a **break** or end-of-file is encountered.

case *word* **in** [*pattern* [|*pattern* ...) *list* **;;**] ... **esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file name generation (see **File Name Generation** below).

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, the **if** command returns a zero exit status.

while *list* **do** *list* **done**
until *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, the **while** command returns a zero exit status; **until** may be used instead of **while** to negate the loop termination test.

(*list*)

Execute *list* in a separate environment. If two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below.

{ *list*;}
: *List* is simply executed. Note that { is a *keyword* and requires a blank to be recognized.

function *identifier* { *list* ;}
identifier () { *list* ;}
: Define a function referenced by *identifier*. The body of the function is the *list* of commands between { and }. (See **Functions** below.)

time *pipeline*
: The *pipeline* is executed and the elapsed time as well as the user and system time are printed on standard error.

The following keywords are only recognized as the first word of a command and not within quotes:

if then else elif fi case esac for while until do done { } function select time

Comments

A word beginning with **#** causes the word and all following characters up to a newline to be ignored.

Aliasing

The first word of each command is replaced by the text of an **alias** if an **alias** for this word has been defined. The first character of an **alias** name can be any nonspecial printable character, but the remaining characters must be the same as they are for a valid *identifier*. The replacement string can contain any valid shell script including the metacharacters listed above. The first word of each command of the replaced text will not be tested for additional **aliases**. If the last character of the **alias** value is a *blank*, the word following the **alias** will also be checked for **alias** substitution. **Aliases** can be used to redefine special built-in commands but cannot be used to redefine the keywords listed above. **Aliases** can be created, listed, and exported with the **alias** command and can be removed with the **unalias** command. Exported **aliases** remain in effect for subshells but must be reinitialized for separate invocations of the shell (see **Invocation** below).

Aliasing is performed when scripts are read, not while they are executed. Therefore, for an **alias** to take effect the **alias** command must be executed before the command that references the alias is read.

Aliases are frequently used as a short-hand for full path names. An option to the aliasing facility allows the value of the **alias** to be automatically set to the full path name of the corresponding command. These **aliases** are called tracked aliases. The value of a tracked alias is defined the first time the corresponding command is looked up and becomes undefined each time the **PATH** variable is reset. These **aliases** remain tracked so that the next subsequent reference will redefine the value. Several tracked aliases are compiled into the shell. The **-h** option of the **set** command causes each command name that is a valid **alias** name to be tracked alias.

The following *exported aliases* are compiled into the shell but can be unset or redefined:

```
false='let 0'
functions='typeset -f'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
hash='alias -t'
```

Tilde Substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~. If it does, the word up to / is checked to see if it matches a user name in the **/etc/passwd** file. If a match is found, the ~ and the matched login name are replaced by the login directory of the matched user. This is called a *tilde* substitution. If no match is found, the original text is unchanged. A ~ by itself or in front of a / is replaced by the value of the **HOME** parameter. A ~ followed by a + or - is replaced by the value of the parameter **PWD** or **OLDPWD**, respectively.

In addition, the value of each *keyword parameter* is checked to see if it begins with a ~ or if a ~ appears after a **:**. In either of these cases a *tilde* substitution is attempted.

Command Substitution

The standard output from a command enclosed in parentheses preceded by a dollar sign (**$()**) or a pair of grave accents (**` `**) may be used as part or all of a word; trailing newlines are removed. In the second (archaic) form, the string between the quotes is processed for special quoting characters before the command is executed. (See **Quoting** below.) The command substitution **$(cat file)** can be replaced by the equivalent but faster **$(<file)**. Command substitution of most special commands that do not redirect input or output is carried out without creating a separate process.

Parameter Substitution

A *parameter* is an *identifier*, one or more digits, or any of the characters *****, **@**, **#**, **?**, **-**, **$**, and **!**. A *named parameter* (a parameter denoted by an identifier) has a *value* and zero or more *attributes*. *Named parameters* can be assigned *values* and *attributes* by using the **typeset** special command. The attributes supported by the shell are described later with the **typeset** special command. Exported parameters pass values and attributes to subshells but only pass values to the environment.

The shell supports a limited one-dimensional array facility. An element of an array parameter is referenced by a *subscript*. A *subscript* is denoted by a **[**, followed by an *arithmetic expression*, followed by a **]** (see **Arithmetic Evaluation** below). The value of all subscripts must be in the range of 0 through 511. Arrays need not be declared. Any reference to a named parameter with a valid subscript is legal and an array will be created if necessary. Referencing an array without a subscript is the same as referencing the first element.

The *value* of a *named parameter* may also be assigned by writing:

name = *value* [*name* = *value*] ...

If the integer attribute **-i** is set for *name*, the *value* is subject to arithmetic evaluation as described below.

Positional parameters (those denoted by a number) may be assigned values with the **set** special command. Parameter **$0** is set from argument zero when the shell is invoked.

The character **$** is used to introduce substitutable *parameters*.

${*parameter***}**
: The value (if any) of the parameter is substituted. The braces are required when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name or when a named parameter is subscripted. If *parameter* is one or more digits, it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If *parameter* is ***** or **@**, all positional parameters, starting with **$1**, are substituted (separated by a field separator character). If an array *identifier* with subscript ***** or **@** is used, the value for each of the elements is substituted (separated by a field separator character).

${#*parameter***}**
: If *parameter* is ***** or **@**, the number of positional parameters is substituted. Otherwise, the length of the *parameter* value is substituted.

${#*identifier***[*]}**
: The number of elements in the array *identifier* is substituted.

${*parameter***:-***word***}**
: If *parameter* is set and is non-null, substitute its value; otherwise, substitute *word*.

${*parameter***:=***word***}**
: If *parameter* is not set or is null, set it to *word*; the parameter's value is then substituted. Positional parameters may not be assigned to in this way.

${*parameter***:?***word***}**
: If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, a standard message is printed.

${*parameter***:+***word***}**
: If *parameter* is set and is non-null, substitute *word*; otherwise, substitute nothing.

${*parameter***#***pattern*}
${*parameter***##***pattern*}
  If the shell *pattern* matches the beginning of the *parameter*'s value, the value of this substitution is the *parameter*'s value with the matched portion deleted; otherwise, the *parameter*'s value is substituted. In the first form, the smallest matching pattern is deleted and in the latter form the largest matching pattern is deleted.

${*parameter***%***pattern*}
${*parameter***%%***pattern*}
  If the shell *pattern* matches the end of the *parameter*'s value, the value of this substitution is the *parameter*'s value with the matched portion deleted; otherwise, the *parameter*'s value is substituted. In the first form, the smallest matching pattern is deleted and in the latter form the largest matching pattern is deleted.

In the above, *word* is not evaluated unless it will be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-$(pwd)}
```

If the colon (**:**) is omitted from the above expressions, the shell only checks whether *parameter* is set.

The shell automatically sets the following parameters:

#	The number of positional parameters in decimal.
-	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last command executed.
$	The process number of this shell.
_	The last argument of the previous command. This parameter is not set for asynchronous commands. This parameter is also used to hold the name of the matching **MAIL** file when checking for mail. Finally, the value of this parameter is set to the full path name of each program the shell invokes and is passed in the *environment*.
!	The process number of the last background command invoked.
PPID	The process number of the parent of the shell.
PWD	The present working directory set by the **cd** command.
OLDPWD	The previous working directory set by the **cd** command.
RANDOM	Each time this parameter is referenced, a random integer is generated. The sequence of random numbers can be initialized by assigning a numeric value to **RANDOM**.

REPLY	This parameter is set by the **select** statement and by the **read** special command when no arguments are supplied.
SECONDS	Each time this parameter is referenced, the number of seconds since shell invocation is returned. If this parameter is assigned a value, the value returned upon reference will be the value assigned plus the number of seconds since the assignment.

The following parameters are used by the shell:

CDPATH	The search path for the *cd* command.
COLUMNS	If this variable is set, the value is used to define the width of the edit window for the shell edit modes and for printing **select** lists.
EDITOR	If the value of this variable ends in **emacs**, **gmacs**, or **vi** and the **VISUAL** variable is not set, the corresponding option will be turned on (see the special command **set** below).
ENV	If this parameter is set, parameter substitution is performed on the value to generate the path name of the script to be executed when the shell is invoked. (See **Invocation** below.) This file is typically used for **alias** and **function** definitions.
FCEDIT	The default editor name for the **fc** command.
IFS	Internal field separators, normally space, tab, and newline, that are used to separate command words which result from command or parameter substitution and for separating words with the special command **read**. The first character of the **IFS** parameter is used to separate arguments for the **$*** substitution. (See **Quoting** below.)
HISTFILE	If this parameter is set when the shell is invoked, the value is the path name of the file that will store the command history. (See **Command Re-entry** below.)
HISTSIZE	If this parameter is set when the shell is invoked, the number of previously entered commands that are accessible by this shell will be greater than or equal to this number. The default is 128.
HOME	The default argument (home directory) for the **cd** command.
LINES	If this variable is set, the value is used to determine the column length for printing **select** lists. Select lists will print vertically until about two-thirds of **LINES** lines is filled.

MAIL	If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of mail arrival in the specified file.
MAILCHECK	This variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds. When the time has elapsed the shell will check before issuing the next prompt.
MAILPATH	A list of file names separated by colons (:). If this parameter is set, the shell informs the user of any modifications to the specified files that have occurred within the last **MAILCHECK** seconds. Each file name can be followed by a ? and a message that will be printed. The message will undergo parameter and command substitution; the name of the file that changed will be substituted for the parameter **$_**. The default message is "you have mail in $_".
PATH	The search path for commands (see **Execution** below). The user may not change **PATH** when executing under *krsh* (except in **.profile**).
PS1	The value of this parameter is expanded for parameter substitution to define the primary prompt string thta is "**$** " by default. The character **!** in the primary prompt string is replaced by the *command* number (see **Command Re-entry** below).
PS2	Secondary prompt string. ("> " by default.)
PS3	Selection prompt string used within a **select** loop. ("#? " by default.)
SHELL	The path name of the shell is kept in the environment. At invocation, if the value of this variable contains an **r** in the basename, the shell becomes restricted.
TMOUT	If set to a value greater than zero, the shell will terminate if a command is not entered within the prescribed number of seconds after issuing the **PS1** prompt. (The shell can be compiled with a maximum bound for this value that cannot be exceeded.)
VISUAL	If the value of this variable ends in **emacs**, **gmacs**, or **vi**, the corresponding option will be turned on (see the special command **set** below).

The shell gives default values to **PATH, PS1, PS2, MAILCHECK, TMOUT** and **IFS**, while the shell does not set **HOME, SHELL, ENV**, and **MAIL** (although *login*(1) sets **HOME**). On some systems *login*(1) also sets **MAIL** and **SHELL**.

Blank Interpretation

After parameter and command substitution, the substitution results are scanned for the field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation

Following substitution, each command *word* is scanned for the characters *, ?, and [unless the -f option has been **set**. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word remains unchanged. When a *pattern* is used for file name generation, the character . at the start of a file name or immediately following a /, and the character /, must be matched explicitly. In other instances of pattern matching, the / and . are not treated specially.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening "[" is an "!", any character not enclosed is matched. A - can be included in the character set by making it the first or last character.

Quoting

Each *metacharacter* listed above has a special meaning to the shell and causes termination of a word unless quoted. A character may be *quoted* (made to stand for itself) by preceding it with a \. The pair **\newline** is ignored. All characters enclosed between a pair of single quote marks (' ') are quoted. A single quote cannot appear within single quotes. Inside double quote marks (" "), parameter and command substitution occurs and \ quotes the characters \, `, ", and **$**. The meanings of **$*** and **$@** are identical when not quoted or when used as a parameter assignment value or file name. However, when used as a command argument, **"$*"** is equivalent to **"$1***d***$2***d*...**"**, where *d* is the first character of the **IFS** parameter; whereas, **"$@"** is equivalent to **"$1" "$2"** .... Inside grave quote marks (` `), \ quotes the characters \, `, and **$**. If the grave quotes occur within double quotes, \ also quotes the character ".

The special meaning of keywords or aliases can be removed by quoting any character of the keyword. The recognition of function names or special command names listed below cannot be altered by quoting them.

Arithmetic Evaluation

An ability to perform integer arithmetic is provided with the special command **let**. Evaluations are performed using *long* arithmetic. Constants have the form [*base#*]*n* where *base* is a decimal number between 2 and 36, representing the arithmetic base, and *n* is a number in that base. If *base* is omitted, base 10 is used.

An internal integer representation of a *named parameter* can be specified with the -i option of the **typeset** special command. When this attribute is selected, the first assignment to the parameter determines the arithmetic base to be used when parameter substitution occurs.

Since many arithmetic operators require quoting, an alternative form of the **let** command is provided. For any command that begins with ((, all characters until a matching)) are treated as a quoted expression. More precisely, ((...)) is equivalent to **let** "...".

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If a newline is typed and further input is needed to complete a command, the secondary prompt (the value of **PS2**) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and not be passed to the invoked command. Command and parameter substitution occurs before *word* or *digit* is used except as noted below. File name generation occurs only if the pattern matches a single file and blank interpretation is not performed.

< *word*	Use file *word* as standard input (file descriptor 0).
> *word*	Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise, it is truncated to zero length.
>> *word*	Use file *word* as standard output. If the file exists, output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
<<[-]*word*	The shell input is read up to a line that is the same as *word*, or to an end-of-file. No parameter substitution, command substitution, or file name generation is performed on *word*. The resulting document, called a *here-document*, becomes the standard input. If any character of *word* is quoted, then no interpretation is placed on the characters of the document; otherwise, parameter and command substitution occurs, **\newline** is ignored, and \ must be used to quote the characters \, $, \`, and the first character of *word*. If - is appended to <<, all leading tabs are stripped from *word* and from the document.
<&*digit*	The standard input is duplicated from file descriptor *digit* (see *dup*(2)). Similarly for the standard output using >&*digit*.

<&- The standard input is closed. Similarly for the standard output using **>&-**.

If one of the above is preceded by a digit, the file descriptor number referred to is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (*file descriptor*, *file*) association at the time of evaluation. For example:

... 1>*fname* 2>&1

first associates file descriptor 1 with file *fname*. It then associates file descriptor 2 with the file associated with file descriptor 1 (*fname*). If the order of redirections is reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be associated with file *fname*.

If a command is followed by **&** and job control is not active, the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for execution contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The *environment* (see *environ*(5)) is a list of name-value pairs that is passed to an executed program as a normal argument list is. The names must be *identifiers* and the values are character strings. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value and marking it *export*. Executed commands inherit the environment. If the user modifies the values of these parameters or creates new ones, using the **export** or **typeset -x** commands they become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions that must be noted in **export** or **typeset -x** commands.

The environment for any *simple-command* or **function** may be augmented by prefixing it with one or more parameter assignments. A parameter assignment argument is a word with the form *identifier=value*. Thus

TERM=450 *cmd args*

and

(export TERM; TERM=450; *cmd args*)

are equivalent (as far as the execution of *cmd* shown above is concerned).

If the **-k** flag is set, *all* parameter assignment arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c**:

```
echo a=b c
set -k
echo a=b c
```

Functions

The **function** keyword, described in the **Commands** section above, defines shell functions. Shell functions are read and stored internally. Alias names are resolved when the function is read. Functions are executed like commands with the arguments passed as positional parameters. (See **Execution** below).

Functions execute in the same process as the caller and share all files, traps (other than **EXIT** and **ERR**), and present working directory with the caller. A trap set on **EXIT** in a function is executed after the function completes. Ordinarily, variables are shared between the calling program and the function. However, the **typeset** special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command **return** is used to return from function calls. Errors within functions return control to the caller.

Function identifiers can be listed with the **-f** option of the **typeset** special command. The text of functions will also be listed. Function can be undefined with the **-f** option of the **unset** special command.

Ordinarily, functions are unset when the shell executes a shell script. The **-xf** option of the **typeset** command allows a function to be exported to scripts executed without a separate invocation of the shell. Functions that need to be defined across separate invocations of the shell should be placed in the **ENV** file.

Jobs

If the **monitor** *option* of the **set** command is turned on, an interactive shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers. When a job is started asynchronously with **&**, the shell prints a line that looks like:

```
[1] 1234
```

indicating that the job started asynchronously was job number 1 and had one (top-level) process, whose process ID was 1234.

This paragraph and the next require features that are not in all versions of UNIX and may not apply. If a job is running and the user wishes to do something else, the key <CONTROL>-Z may be hit, which sends a STOP signal to the current job. The shell will then normally indicate that the job has been "Stopped" and print another prompt. The state of this job can then be manipulated, putting it in the background with the **bg** command, or run some other commands and then eventually bring the job back to the foreground with the foreground command **fg**. A <CONTROL>-Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command **stty tostop**. If this tty option is set, background jobs will stop when they try to produce output as they do when they try to read input.

There are several ways to refer to jobs in the shell. The character **%** introduces a job name. To refer to job number 1, it can be named as **%1**. Jobs can also be named by prefixes of the string typed in to **kill** or restart them. Thus, on systems that support job control, **fg %ed** would normally restart a suspended *ed*(1) job if a suspended job whose name began with the string "ed" existed.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a + and the previous job with a -. The abbreviation **%+** refers to the current job and **%-** refers to the previous job. **%%** is also a synonym for the current job.

This shell learns immediately when a process changes state. It normally informs the user when a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that the user's work is not otherwise disturbed.

When trying to leave the shell while jobs are running or stopped, the user will be warned that "You have stopped(running) jobs." The **jobs** command can then be used to see what they are. If this is done or exiting is immediately tried again, the shell will not give a second warning, and the stopped jobs will be terminated.

Signals

The INT and QUIT signals for an invoked command are ignored if the command is followed by **&** and job **monitor** *option* is not active. Otherwise, signals have the values inherited by the shell from its parent (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions occur. If the command name matches one of the special commands listed below, it is executed within the current shell process. Next, the command name is checked to see if it matches one of the user defined functions. If it does, the positional parameters are saved and then reset to the arguments of the *function* call. When the *function* completes or issues a **return**, the positional parameter list is restored and any trap set on **EXIT** within the function is executed. The value of a *function* is the value of the last command executed. A function is also executed in the current shell process. If a command name is not a special command or a user-defined *function*, a process is created and an attempt is made to execute the command using *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (**:**). The default path is **/bin:/usr/bin:** (specifying **/bin**, **/usr/bin**, and the current directory in that order). The current directory can be specified by two or more adjacent colons or by a colon at the beginning or end of the path list. If the command name contains a **/**, the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. All nonexported aliases, functions, and named parameters are removed in this case. If the shell command file does not have read permission or if the setuid and/or setgid bits are set on the file, the shell executes an agent who sets up the permissions and executes the shell with the shell command file passed as an open file. A parenthesized command is also executed in a subshell without removing nonexported quantities.

Command Re-entry

The text of the last **HISTSIZE** (default 128) commands entered from a terminal device is saved in a *history* file. The file **$HOME/.sh_history** is used if the **HISTFILE** variable is not set or is not writable. A shell can access the commands of all *interactive* shells that use the same named **HISTFILE**. The special command **fc** is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first characters of the command. A single command or range of commands can be specified. If an editor program is not specified as an argument to **fc**, the value of the parameter **FCEDIT** is used. If **FCEDIT** is not defined, **/bin/ed** is used. The edited command(s) is printed and re-executed on leaving the editor. The editor name - is used to skip the editing phase and to re-execute the command. In this case, a substitution parameter with the form *old***=***new* can be used to modify the command before execution. For example, if **r** is aliased to **'fc -e -'**, typing **r bad=good c** will re-execute the most recent command that starts with **c**, replacing the first occurrence of the string "bad" with the string "good".

In-line Editing Options

Normally, each command line entered from a terminal device is simply typed followed by a newline (<RETURN> or <LINE FEED>). If either the **emacs**, **gmacs**, or **vi** *option* is active, the user can edit the command line. To be in either of these edit modes, **set** the corresponding *option*. An editing *option* is automatically selected each time the **VISUAL** or **EDITOR** variable is assigned a value ending in either of these *option* names.

The editing features require that the user's terminal accept <RETURN> as a carriage return without a line feed and that a space (" ") must overwrite the current character on the screen. ADM terminal users should set the "space - advance" switch to "space". Hewlett-Packard® series 2621 terminal users should set the straps to "bcGHxZ etX".

The editing modes implement a concept where the user is looking through a window at the current line. The window width is the value of **COLUMNS** if it is defined; otherwise it is 80. If the line is longer than the window width minus two, a mark is displayed at the end of the window to notify the user. As the cursor moves and reaches the window boundaries, the window will be centered about the cursor. The mark is a **>** (**<**, *****) if the line extends on the right (left, both) side(s) of the window.

Emacs Editing Mode

This mode is entered by enabling either the **emacs** or **gmacs** *option*. The only difference between these two modes is the way they handle **^T**. To edit, the user moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. All editing commands are control characters or escape sequences. The notation for control characters is caret (**^**) followed by the character. For example, **^F** is the notation for <CONTROL>-F. This is entered by pressing "f" while holding down the <CONTROL> key. The <SHIFT> key is *not* pressed. (The notation **^?** indicates the <DEL> (delete) key.)

The notation for escape sequences is **M-** followed by a character. For example, **M-f** (pronounced Meta f) is entered by pressing <ESC> (ASCII 033) followed by "f". (**M-F** would be the notation for <ESC> followed by <SHIFT> (capital) "F".)

All edit commands operate from any place on the line (not just at the beginning). Neither the <RETURN> nor the <LINE FEED> key is entered after edit commands except when noted.

^F	Move cursor forward (right) one character.
M-f	Move cursor forward one word. (The editor interprets a word as a string of characters consisting of only letters, digits and underscores.)
^B	Move cursor backward (left) one character.
M-b	Move cursor backward one word.
^A	Move cursor to start of line.
^E	Move cursor to end of line.
^]*char*	Move cursor to character *char* on current line.
^X^X	Interchange cursor and mark.
erase	(User-defined erase character as defined by the *stty*(1) command, usually **^H** or **#**.) Delete previous character.
^D	Delete current character.
M-d	Delete current word.
M-^H	(Meta-backspace) Delete previous word.
M-h	Delete previous word.
M-^?	(Meta-<DEL>) Delete previous word (if the interrupt character is **^?** (<DEL>, the default), this command will not work).
^T	Transpose current character with next character in **emacs** mode. Transpose two previous characters in **gmacs** mode.
^C	Capitalize current character.
M-c	Capitalize current word.
M-l	Change the current word to lowercase.

^K	Kill from the cursor to the end of the line. If given a parameter of zero, kill from the start of line to the cursor.
^W	Kill from the cursor to the mark.
M-p	Push the region from cursor to mark on the stack.
kill	(User-defined kill character as defined by the *stty*(1) command, usually **^G** or **@**.) Kill the entire current line. If two *kill* characters are entered in succession, all kill characters from then on cause a line feed (useful when using paper terminals).
^Y	Restore last item removed from line. (Yank item back to the line.)
^L	Line feed and print current line.
^@	(Null character) Set mark.
M-	(Meta space) Set mark.
^J	(<LINE FEED>) Execute current line.
^M	(<RETURN>) Execute current line.
eof	End-of-file character, normally **^D**, will terminate shell if current line is null.
^P	Fetch previous command. Each time **^P** is entered, the previous command is accessed.
M-<	Fetch least recent (oldest) history line.
M->	Fetch most recent (youngest) history line.
^N	Fetch next command. Each time **^N** is entered the next command is accessed.
^R*string*	Reverse search history for a previous command line containing *string*. If a parameter of zero is given, the search is forward. *String* is terminated by a <RETURN> or <LINE FEED>. If *string* is omitted, the next command line containing the most recent *string* is accessed. In this case, a parameter of zero reverses the direction of the search.
^O	Operate - Execute the current line and fetch the next line relative to it from the history file.
M-*digits*	(<ESC>) Define numeric parameter. The digits are interpreted as a parameter to the next command. The commands that accept a parameter are **.**, **^F**, **^B**, *erase*, **^D**, **^K**, **^R**, **^P**, **^N**, **M-.**, **M-_**, **M-b**, **M-c**, **M-d**, **M-f**, **M-h**, and **M-^H**.
M-*letter*	Soft-key - The alias list is searched for an **alias** with the name *_letter*; if an alias with this name is defined, its value will be inserted on the input queue. The *letter* must not be one of the above meta-functions.
M-.	The last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.
M-_	Same as **M-.**.
M-*	Attempt file name generation on the current word. An asterisk is appended if the word does not contain any special pattern characters.

M-<ESC>	Same as **M-***.
M-=	List files matching current word pattern if an asterisk was appended.
^U	Multiply parameter of next command by 4.
\	Escape next character. Editing characters, the user's erase, kill, and interrupt (normally **^?**) characters may be entered in a command line or in a search string if preceded by a \. The \ removes the next character's editing features (if any).
^V	Display version of the shell.

Vi Editing Mode

There are two typing modes. Initially, when a command is entered the user is in the *input* mode. To edit, the user enters *control* mode by typing <ESC> (ASCII 033) and moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. Most control commands accept an optional repeat *count* before the command.

When in vi mode on most systems, canonical processing is initially enabled and the command will be echoed again if the speed is 1200 baud or greater and it contains any control characters or less than one second has elapsed since the prompt printed. The <ESC> character terminates canonical processing for the remainder of the command and the user can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.

If the *option* **viraw** is also set, the terminal's canonical processing will always be disabled. This mode is implicit for systems that do not support two alternate end-of-line delimiters and may be helpful for certain terminals.

Input Edit Commands

By default, the editor is in input mode.

erase	(User-defined erase character as defined by the *stty*(1) command, usually **^H** or **#**.) Delete previous character.
^W	Delete the previous blank separated word.
^D	Terminate the shell.
^V	Escape next character. Editing characters, the user's erase or kill characters may be entered in a command line or in a search string if preceded by a **^V**. The **^V** removes the next character's editing features (if any).
\	Escape the next *erase* or *kill* character.

Motion Edit Commands

These commands will move the cursor.

[*count*]**l**	Cursor forward (right) one character.
[*count*]**w**	Cursor forward one alphanumeric word.
[*count*]**W**	Cursor to beginning of next word that follows a blank.
[*count*]**e**	Cursor to end of word.
[*count*]**E**	Cursor to end of current blank-delimited word.

[*count*]**h**	Cursor backward (left) one character.
[*count*]**b**	Cursor backward one word.
[*count*]**B**	Cursor to preceding blank-separated word.
[*count*]**f***c*	Find next character *c* in current line.
[*count*]**F***c*	Find the previous character *c* in the current line.
[*count*]**t***c*	Equivalent to **f** followed by **h**.
[*count*]**T***c*	Equivalent to **F** followed by **l**.
;	Repeats the last single-character find command, **f**, **F**, **t**, or **T**.
,	Reverses the last single-character find command.
0	Cursor to start of line.
^	Cursor to first nonblank character in line.
$	Cursor to end of line.

Search Edit Commands

These commands access the command history.

[*count*]**k**	Fetch previous command. Each time **k** is entered, the previous command is accessed.
[*count*]**-**	Equivalent to **k**.
[*count*]**j**	Fetch next command. Each time **j** is entered, the next command is accessed.
[*count*]**+**	Equivalent to **j**.
[*count*]**G**	The command number *count* is fetched. The default is the least recent history command.
/*string*	Search backward through history for a previous command containing *string*. *String* is terminated by a <RETURN> or <LINE FEED>. If *string* is null, the previous string will be used.
?*string*	Same as / except that search will be forward.
n	Search for next match of the last pattern to / or ? commands.
N	Search for next match of the last pattern to / or ?, but in reverse direction. Search history for the *string* entered by the previous / command.

Text Modification Edit Commands

These commands will modify the line.

a	Enter input mode and enter text after the current character.
A	Append text to the end of line. Equivalent to **$a**.
[*count*]**c***motion*	
c[*count*]*motion*	Delete current character through the character that *motion* would move the cursor to and enter input mode. If *motion* is **c**, the line will be deleted and input mode entered.
C	Delete the current character through end of line and enter input mode. Equivalent to **c$**.

S	Equivalent to **cc**.
D	Delete the current character through the end of line. Equivalent to **d$**.
[*count*]**d***motion*	
d[*count*]*motion*	Delete current character through the character that *motion* would move to. If *motion* is **d**, the line will be deleted.
i	Enter input mode and insert text before the current character.
I	Insert text before beginning of line. Equivalent to the two-character sequence **^i**.
[*count*]**P**	Place the previous text modification before the cursor.
[*count*]**p**	Place the previous text modification after the cursor.
R	Enter input mode and replace characters on the screen with characters typed overlaying them.
r*c*	Replace the current character with *c*.
[*count*]**x**	Delete current character.
[*count*]**X**	Delete preceding character.
[*count*]**.**	Repeat previous text modification command.
~	Invert case of the current character and advance cursor.
[*count*]**_**	Causes the *count* word of the previous command to be appended and input mode entered. The last word is used if *count* is omitted.
*****	Causes an ***** to be appended to the current word and file name generation attempted. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.

Other Edit Commands

Miscellaneous commands.

[*count*]**y***motion*	
y[*count*]*motion*	Yank current character through character that *motion* would move the cursor to and put them in the delete buffer. The text and cursor are unchanged.
Y	Yanks from current position to end of line. Equivalent to **y$**.
u	Undo the last text modifying command.
U	Undo all the text modifying commands performed on the line.
[*count*]**v**	Returns the command **fc -e {VISUAL:-${EDITOR:-vi}}** *count* in the input buffer. If *count* is omitted, the current line is used.
^L	Line feed and print current line. Works only in control mode.

^J	(<LINE FEED>) Execute the current line, regardless of mode.
^M	(<RETURN>) Execute the current line, regardless of mode.
#	Sends the line after inserting **#** in front of line and after each newline. Useful for causing the current line to be inserted in the history without being executed.
=	List the file names that match the current word with an asterisk appended it.
@*letter*	The alias list is searched for an alias with the name _*letter* and if an alias with this name is defined, its value will be inserted on the input queue for processing.

Special Commands

The following simple-commands are executed in the shell process. Input/output redirection is permitted. Unless otherwise indicated, the output is written on file descriptor 1. Commands preceded by one or two † are treated specially in the following ways:

1. Parameter assignment lists preceding the command remain in effect when the command completes.
2. They are executed in a separate process when used in command substitution.
3. Errors in commands preceded by †† cause the script that contains them to abort.

† : [*arg* ...]

The command only expands parameters. A zero exit code is returned.

†† . *file* [*arg* ...]

Read and execute commands from *file* and return. The commands are executed in the current shell environment. The search path specified by **PATH** is used to find the directory containing *file*. If any arguments *arg* are given, they become the positional parameters. Otherwise, the positional parameters are unchanged.

alias [**-tx**] [*name*[=*value*] ...]

Alias with no arguments prints the list of aliases in the form *name=value* on standard output. An **alias** is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for **alias** substitution. The **-t** flag is used to set and list tracked aliases. The value of a tracked alias is the full path name corresponding to the given *name*. The value becomes undefined when the value of **PATH** is reset but the aliases remained tracked. Without the **-t** flag, for each *name* in the argument list with no *value* given, the name and value of the alias is printed. The **-x** flag sets or prints exported aliases. An exported alias is defined across subshell environments. **Alias** returns true unless a *name* is given for which no alias has been defined.

bg [**%***job*]
: This command is only built-in on systems that support job control. It puts the specified *job* in the background. The current job is put in the background if *job* is not specified.

break [*n*]
: Exit from the enclosing **for**, **while**, **until**, or **select** loop, if any. If *n* is specified, break *n* levels.

continue [*n*]
: Resume the next iteration of the enclosing **for**, **while**, **until**, or **select** loop. If *n* is specified, resume at the *n*th enclosing loop.

† **cd** [*arg*]
† **cd** *old new*
: This command can be in either of two forms. In the first form it changes the current directory to *arg*. If *arg* is -, the directory is changed to the previous directory. The shell parameter **HOME** is the default *arg*. The parameter PWD is set to the current directory. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a /, the search path is not used. Otherwise, each directory in the path is searched for *arg*.

  The second form of **cd** substitutes the string *new* for the string *old* in the current directory name, PWD, and tries to change to this new directory.

  The **cd** command may not be executed by *krsh*.

echo [*arg* ...]
: See *echo*(1) for usage and description.

†† **eval** [*arg* ...]
: The arguments are read as input to the shell and the resulting command(s) executed.

†† **exec** [*arg* ...]
: If *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and affect the current process. If no arguments are given, this command modifies file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 opened with this mechanism are closed when invoking another program.

exit [*n*]
: Causes the shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. An

end-of-file will also cause the shell to exit unless the shell's **ignoreeof** *option* is on (see **set** below). on.

†† **export** [*name* ...]
: The given *names* are marked for automatic export to the *environment* of subsequently-executed commands.

†† **fc** [**-e** *ename*] [**-nlr**] [*first*] [*last*]
†† **fc -e** - [*old*=*new*] [*command*]
: In the first form, a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the flag -l is selected, the commands are listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, the value of the parameter **FCEDIT** (default **/bin/ed**) is used as the editor. When editing is complete, the edited command(s) is executed. If *last* is not specified, it will be set to *first*. If *first* is not specified, the default is the previous command for editing and -16 for listing. The flag **-r** reverses the order of the commands and the flag **-n** suppresses command numbers when listing. In the second form, the *command* is re-executed after *old*=*new* is substituted.

fg [%*job*]
: This command is only built-in on systems that support job control. If *job* is specified, the command brings it to the foreground. Otherwise, the current job is brought to the foreground.

jobs [-l]
: Lists the active jobs. The -l option lists process id's in addition to the normal information.

kill [-*sig*] *process* ...
: Sends either the TERM (terminate) signal or the signal *sig* to the specified jobs or processes. Signals are given by number or by names (as given in **/usr/include/signal.h**, stripped of the prefix "SIG"). The signal numbers and names are listed by **kill** -l. If the signal being sent is TERM (terminate) or HUP (hangup), the job or process will be sent a CONT (continue) signal if it is stopped. The argument *process* can be either a process ID or a job.

let *arg* ...
: Each *arg* is an *arithmetic expression* to be evaluated. All calculations are performed as long integers and overflow is not checked. Expressions consist of constants, named parameters, and operators. The following set of operators, listed in order of decreasing precedence, have been implemented:

—	unary minus

!	logical negation
* / %	multiplication, division, remainder
+ −	addition, subtraction
<= >= < >	comparison
== !=	equality, inequality
=	arithmetic replacement

Subexpressions in parentheses () are evaluated first and can be used to override the above precedence rules. The evaluation within a precedence group is from right to left for the = operator and from left to right for the others.

A parameter name must be a valid *identifier*. When a parameter is encountered, the value associated with the parameter name is substituted and expression evaluation resumes. Up to nine levels of recursion are permitted.

The return code is 0 if the value of the last expression is nonzero, and 1 otherwise.

†† **newgrp** [*arg* ...]
Equivalent to **exec newgrp** *arg* ....

print [-**Rnprsu**[*n*]] [*arg* ...]
The shell output mechanism. With no flags or with flag -, the arguments are printed on standard output as described by *echo*(1). In raw mode, -**R** or -**r**, the escape conventions of *echo*(1) are ignored. The -**R** option will print all subsequent arguments and options other than -**n**. The -**p** option causes the arguments to be written to the pipe of the process spawned with **|&** instead of standard output. The -**s** option causes the arguments to be written to the history file instead of standard output. The -**u** flag can be used to specify a one-digit file descriptor unit number *n* on which the output will be placed. The default is 1. If the flag -**n** is used, no newline is added to the output.

pwd Equivalent to **print** -**r** - **$PWD**.

read [-**prsu**[*n*]] [*name?prompt*] [*name* ...]
The shell input mechanism. One line is read and is broken into words using the characters in **IFS** as separators. In raw mode, -**r**, a \ at the end of a line does not signify line continuation. The first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The -**p** option causes the input line to be taken from the input pipe of a process spawned by the shell using **|&**. If the -**s** flag is present, the input will be saved as a command in the history file. The flag -**u** can specify a one-digit file descriptor unit to read from. The file descriptor can be opened with the **exec** special command. The default value of *n* is 0. If *name* is omitted, **REPLY** is the default *name*. The return code is 0 unless an end-of-file is encountered. An end-of-file with the -**p** option causes cleanup for this process so that another can be

spawned. If the first argument contains a ?, the remainder of this word is used as a *prompt* when the shell is interactive. If the given file descriptor is open for writing and is a terminal device, the prompt is placed on this unit. Otherwise the prompt is issued on file descriptor 2. The return code is 0 unless an end-of-file is encountered.

†† **readonly** [*name* ...]
: The given *names* are marked read-only and these names cannot be changed by subsequent assignment.

†† **return** [*n*]
: Causes a shell **function** to return to the invoking script with the return status specified by *n*. If *n* is omitted, the return status is that of the last command executed. If **return** is invoked while not in a **function** or a **.** script, it is the same as an **exit**.

set [**-aefhkmnostuvx**] [**-o** *option* ...] [*arg* ...]
: The flags for this command are defined as follows:

-a All subsequent parameters that are defined are automatically exported.

-e If the shell is noninteractive and a command fails, execute the **ERR** trap, if set, and exit immediately. This mode is disabled while reading profiles.

-f Disables file name generation.

-h Each command whose name is an *identifier* becomes a tracked alias when first encountered.

-k All parameter assignment arguments, not just those that precede the command name, are placed in the environment for a command.

-m Background jobs will run in a separate process group and a line will print on completion. The exit status of background jobs is reported in a completion message. This flag is turned on automatically for interactive shells.

-n Read commands, but do not execute them. Ignored for interactive shells.

-o The following argument can be one of the following *option* names:

allexport	Same as **-a**.
errexit	Same as **-e**.
bgnice	All background jobs run at a lower priority.
emacs	Puts *ksh* in an *emacs*-style in-line editor for command entry.
gmacs	Puts *ksh* in a *gmacs*-style in-line editor for command entry.
ignoreeof	The shell will not exit on end-of-file. The command **exit** must be used.
keyword	Same as **-k**.

markdirs	All directory names resulting from file name generation have a trailing / appended.
monitor	Same as **-m**.
noexec	Same as **-n**.
noglob	Same as **-f**.
nounset	Same as **-u**.
protected	Same as **-p**.
verbose	Same as **-v**.
trackall	Same as **-h**.
vi	Puts *ksh* in insert mode of a *vi*-style in-line editor until <ESC> is pressed. This puts *ksh* in move mode. A <RETURN> executes the line.
viraw	Each character is processed as it is typed in *vi* mode.
xtrace	Same as **-x**.

If no *option* name is supplied, the current *option* settings are printed.

-p	Resets the **PATH** variable to the default value, disables processing of the **$HOME/.profile** file, and uses the file **/etc/suid_profile** instead of the **ENV** file. This mode is automatically enabled when the effective UID (GID) does not equal the real UID (GID).
-s	Sort the positional parameters.
-t	Exit after reading and executing one command.
-u	Treat unset parameters as an error when substituting.
-v	Print shell input lines as they are read.
-x	Print commands and their arguments as they are executed.
-	Turns off **-x** and **-v** flags and stops examining arguments for flags.
--	Do not change any flags; this is useful in setting **$1** to a value beginning with -. If no arguments follow this flag, the positional parameters are unset.

Using + rather than - turns these flags off. These flags can also be used when the shell is invoked. The current setting of flags is found in **$-**. The remaining arguments are positional parameters and are assigned, in order, to **$1 $2** .... If no arguments are given, the values of all names are printed on the standard output.

† **shift** [*n*]

The positional parameters from **$***n*+1 ... are renamed **$1** .... The default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a non-negative number less than or equal to **$#**.

test [*expr*]

Evaluate conditional expression *expr* (see *test*(1) for use and description). The arithmetic comparison operators are not restricted to integers. They allow any arithmetic expression. Four additional primitive expressions are allowed:

-L *file*	True if *file* is a symbolic link.
file1 **-nt** *file2*	True if *file1* is newer than *file2*.
file1 **-ot** *file2*	True if *file1* is older than *file2*.
file1 **-ef** *file2*	True if *file1* has the same device and i-node number as *file2*.

times Print the accumulated user and system times for the shell and for processes run from the shell.

trap [*arg*] [*sig*] ...
Arg is a command to be read and executed when the shell receives signal(s) *sig*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be given as a number or as the name of the signal. Trap commands are executed in the order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. If *arg* is omitted or is -, all trap(s) *sig* are reset to their original values. If *arg* is the null string, the shell and the commands the shell invokes ignore *sig*. If *sig* is **ERR**, *arg* will be executed when a command has a nonzero exit code. This trap is not inherited by functions. If *sig* is **0** or **EXIT** and the **trap** statement is executed in the body of a function, the command *arg* is executed after the function completes. If *sig* is **0** or **EXIT** for a **trap** set outside any function, the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

†† **typeset** [**-HLRZfilprtux**[*n*] [*name*[**=***value*]] ...]
When invoked inside a function, a new instance of the parameter *name* is created. The parameter value and type are restored when the function completes. The following list of attributes may be specified:

-H This flag provides CLIX to host name file mapping on non-CLIX machines.

-L Left justify and remove leading blanks from *value*. If *n* is nonzero it defines the width of the field. Otherwise, it is determined by the width of the value of the first assignment. When a value is assigned to *name*, it is filled on the right with blanks or truncated if necessary to fit in the field. Leading zeros are removed if the **-Z** flag is also set. The **-R** flag is turned off.

-R Right justify and fill with leading blanks. If *n* is nonzero it defines the width of the field. Otherwise, it is determined by the width of the value of the first assignment. When a value is assigned to *name*, it is filled on the left with blanks or truncated from the end if necessary to fit in the field. The **-L** flag is turned off.

-Z Right justify and fill with leading zeros if the first nonblank character is a digit and the **-L** flag was not been. If *n* is nonzero it defines the width of the field. Otherwise, it is determined by the width of the value of first assignment.

-f The names are function names rather than parameter names. No assignments can be made and the only other valid flags are **-t**, which turns on execution tracing for this function, and **-x**, to allow the function to remain in effect across shell procedures executed in the same process environment.

-i Parameter is an integer. This makes arithmetic faster. If *n* is nonzero it defines the output arithmetic base. Otherwise, the first assignment determines the output base.

-l All uppercase characters are converted to lowercase. The uppercase flag **-u** is turned off.

-p The output of this command, if any, is written on the two-way pipe

-r The given *names* are marked as read-only and cannot be changed by subsequent assignment.

-t Tags the named parameters. Tags are user-definable and have no special meaning to the shell.

-u All lowercase characters are converted to uppercase. The lowercase flag **-l** is turned off.

-x The given *names* are marked for automatic export to the *environment* of subsequently executed commands.

Using **+** rather than **-** turns these flags off. If no *name* arguments are given but flags are specified, a list of *names* (and optionally the *values*) of the *parameters* that have these flags set is printed. (Using **+** rather than **-** keeps the values to be printed.) If no *names* or flags are given, the *names* and *attributes* of all *parameters* are printed.

ulimit [*n*]
Imposes a size limit of *n* 512 byte blocks on files written by child processes. (Files of any size may be read.) If *n* is not given, the current limit is printed.

umask [*nnn*]
The user's file-creation mask is set to *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

unalias *name ...*
The *parameters* given by the list of *names* are removed from the *alias* list.

unset [**-f**] *name ...*
The parameters given by the list of *names* are unassigned. (Their values and attributes are erased.) Read-only variables cannot be unset. If the flag **-f** is set, the names refer to **function** names.

wait [*n*]
Wait for the specified child process and report its termination status. If *n* is not given, all currently active child processes are waited for. The return code from this command is that of the process waited for.

whence [**-v**] *name* ...
For each *name*, indicate how it would be interpreted if used as a command name. The flag **-v** produces a more verbose report.

Invocation

If the shell is invoked by *exec*(2) and the first character of argument zero (**$0**) is **-**, the shell is assumed to be a *login* shell and commands are read from **/etc/profile** and then from either **.profile** in the current directory or **$HOME/.profile** (if either file exists). Next, commands are read from the file named by performing parameter substitution on the value of the environment parameter **ENV** if the file exists. If the **-s** flag is not present and *arg* is, a path search is performed on the first *arg* to determine the name of the script to execute. The script *arg* must have read permission and any setuid and getgid settings will be ignored. Commands are then read as described below; the following flags are interpreted by the shell when it is invoked:

-c *string* If the **-c** flag is present, commands are read from *string*.

-s If the **-s** flag is present or if no arguments remain, commands are read from the standard input. Shell output, except for the output of the special commands listed above, is written to file descriptor 2.

-i If the **-i** flag is present or if the shell input and output are attached to a terminal (as told by *ioctl*(2)), this shell is *interactive*. In this case, TERM is ignored (so that **kill 0** does not kill an interactive shell) and INTR is caught and ignored (so that **wait** is interruptible). In all cases, the shell ignores QUIT.

-r If the **-r** flag is present, the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

krsh Only

krsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *krsh* are identical to those of *ksh*, except that the following are disallowed:

Changing directory (see *cd*(1))
Setting the value of **SHELL**, **ENV**, or **PATH**
Specifying path or command names containing **/**
Redirecting output (**>** and **>>**)

The restrictions above are enforced after **.profile** and the **ENV** files are interpreted.

When a command to be executed is a shell procedure, *krsh* invokes *ksh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell while imposing a limited menu of commands. This scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** controls user actions completely, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (such as **/usr/rbin**) that can be safely invoked by *krsh*. Some systems also provide a restricted editor *red*.

FILES

/etc/passwd
/etc/profile
/etc/suid_profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

test(1), signal(2), a.out(4).
cat(1), cd(1), echo(1), env(1), newgrp(1), umask(1), vi(1) in the *UNIX System V User's Reference Manual*.
dup(2), exec(2), fork(2), ioctl(2), lseek(2), pipe(2), umask(2), ulimit(2), wait(2), rand(3C), profile(5) in the *UNIX System V Programmer's Reference Manual*.
environ(5) in the *UNIX System V System Administrator's Reference Manual*.

DIAGNOSTICS

Errors, such as syntax errors, detected by the shell cause the shell to return a nonzero exit status. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above). If the shell is being used noninteractively, shell file execution is abandoned. Runtime errors detected by the shell are reported by printing the command or function name and error condition. If the line number that the error occurred on is greater than one, the line number is also printed in square brackets ([]) after the command or function name.

CAVEATS

If a command that is a "tracked alias" is executed, and then a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **-t** option of the **alias** command to correct this situation.

Some old shell scripts contain a ^ as a synonym for the pipe character **|**.

If a command is piped to a shell command, all variables set in the shell command are lost when the command completes.

Using the **fc** built-in command within a compound command will cause the whole command to disappear from the history file.

The built-in command **.** *file* reads the whole file before any commands are executed. Therefore, **alias** and **unalias** commands in the file will not apply to any functions defined in the file.

NAME

ld - link editor for common object files

SYNOPSIS

ld [*option* ...] *file-name*

DESCRIPTION

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

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

The following options are recognized by *ld*:

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

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

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

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

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

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

-a Create an absolute file. This is the default if the **-r** option is not used. Used with the **-r** *option*, **-a** allocates memory for common symbols.

-s Strip line number entries and symbol table information from the output object file.

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

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

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

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

-Ct *cm* Set the cache mode for the text region to *cm*, where *cm* is one of the following:

pw	Private, write through.
sw	Shared, write through.
cb	Private, copy back.
nc	Noncached.
df	Default for the region. Defaults are defined in the kernel as **sw** for the text region and **cb** for stack and data. (NOTE: These are also the defaults assumed by *ld* in absence of any cache mode directives.)

-Cd *cm* Set the cache mode for the data region to *cm*.

-Cs *cm* Set the cache mode for the stack region to *cm*.

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

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

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

-V	Output a message giving information about the version of *ld* being used.
-VS *num*	Use *num* as a decimal version stamp identifying the **a.out** file produced. The version stamp is stored in the optional header.
-Y [**LU**],*dir*	Change the default directory used for finding libraries. If **L** is specified, the first default directory that *ld* searches, LIBDIR, is replaced by *dir*. If **U** is specified and *ld* has been built with a second default directory, LLIBDIR, that directory is replaced by *dir*. If *ld* was built with only one default directory and **U** is specified, a warning is printed and the option is ignored.

FILES

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

SEE ALSO

as(1), cc(1), mkshlib(1), exit(2), a.out(4).
end(3C), ar(4) in the *UNIX System V Programmer's Reference Manual.*
"Link Editor Directives" and "Shared Libraries" in the *UNIX System V Programmer's Guide.*

CAVEATS

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

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

— When the link editor is called through *cc*(1), a startup routine is linked with the user's program. This routine calls *exit*(2) after execution of the main program. If the user calls the link editor directly, the user must ensure that the program always calls *exit*(2) rather than falling through the end of the entry routine.

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

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

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

NAME

ln - link files

SYNOPSIS

ln [**-f**] [**-s**] *file* ... *target*

DESCRIPTION

ln links *file* ... to *target*. *File* and *target* can never be the same. If *target* is a directory, one or more files are linked to that directory. If *target* is a file, its contents are destroyed.

By default, *ln* makes hard links. A hard link to a file cannot be distinguished from the original directory entry; any changes to a file are effective independently from the name used to reference the file. Hard links cannot span file systems and may not refer to directories.

The **-s** option causes *ln* to create symbolic links. A symbolic link contains the name of the file to which it is linked. The referenced file is used when an *open*(2) operation is performed on the link. A *stat*(2) on a symbolic link will return the file that was linked to; an *lstat*(2) must be executed to obtain information about the link. The *readlink*(2) call may be used to read the contents of a symbolic link. Symbolic links may span file systems and refer to directories. If a symbolic link is made to a file and the file is removed, the link remains and is invalid.

If *ln* determines that the mode of *target* forbids writing, it displays the mode (see *chmod*(2)), prompts for a response, and reads from standard input for one line. If the line begins with **y**, the link occurs if it is permissible. If not, the command exits. When the **-f** option is used or if the standard input is not a terminal, no questions are asked and the link is executed.

SEE ALSO

chmod(1), rm(1).
cp(1) in the *UNIX System V User's Reference Manual.*

WARNINGS

ln will not create hard links across file systems.

Shell metacharacters should be used carefully.

NAME

lpq - BSD spool queue examination program

SYNOPSIS

lpq [+[*n*]] [-l] [**-P** *printer*] [*job* ...] [*user* ...]

DESCRIPTION

lpq examines the spooling area used by *lpd*(1M) for printing files on the line printer and reports the status of the specified jobs or all jobs associated with a user. *lpq* invoked without any arguments reports on any jobs currently in the queue. A -P option may be used to specify a particular *printer*. Otherwise, the default line printer is used (or the value of PRINTER in the environment). If a + argument is supplied, *lpq* displays the spool queue until it empties. Supplying a number immediately after the + sign indicates that *lpq* should sleep *n* seconds between queue scans. All other arguments supplied are interpreted as *user* names or *job* numbers to filter only jobs of interest.

For each job submitted (invocation of *lpr*(1)) *lpq* reports the user's name, current rank in the queue, the names of files composing the job, the job identifier (a number that may be supplied to *lprm*(1) for removing a specific job), and the total size in bytes. The -l option prints information about each of the files composing the job. Normally, only the amount of information that will fit on one line is displayed. Job ordering depends on the algorithm used to scan the spooling directory and is supposed to be first in first out (FIFO). File names composing a job may be unavailable (when *lpr*(1) is used as a sink in a pipeline). In this case, the file is indicated as "(standard input)".

If *lpq* warns that no daemon is present (e.g., due to a malfunction), the *lpc*(1M) command can be used to restart the printer daemon.

FILES

/etc/terminfo	for manipulating the screen for repeated display
/etc/printcap	to determine printer characteristics
/usr/spool/*	the spooling directory, as determined from printcap
/usr/spool/*/cf*	control files specifying jobs
/usr/spool/*/lock	the lock file to obtain the currently-active job

SEE ALSO

lpr(1), lprm(1).
lpc(1M), lpd(1M) in the *CLIX Programmer's & User's Reference Manual.*

DIAGNOSTICS

lpq may be unable to open various files, have the lock file be malformed, or produce garbage files when no daemon is active but files are in the spooling directory.

BUGS

Due to the dynamic nature of the information in the spooling directory, *lpq* may report unreliably. Output formatting is sensitive to the line length of the terminal. This can result in widely-spaced columns.

NAME

lpr - BSD offline print

SYNOPSIS

lpr [**-P** *printer*] [**-#** *num*] [**-C** *class*] [**-J** *job*] [**-T** *title*] [**-i** [*numcols*]] [**-w** *num*] [**-pltngvfrmhs**] [*name* ...]

DESCRIPTION

lpr uses a spooling daemon to print the named files when facilities become available. If no names appear, the standard input is assumed. The **-P** option may be used to force output to a specific *printer*. Normally, the default printer is used (site dependent), or the value of the environment variable PRINTER is used.

The following single letter options are used to notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

-p Use *pr*(1) to format the files.

-l Use a filter that allows control characters to be printed and suppresses page breaks.

-t The files are assumed to contain data from *troff* (cat phototypesetter commands).

-n The files are assumed to contain data from *ditroff* (device-independent *troff*).

-g The files are assumed to contain standard plot data as produced by the *plot*(3X) routines.

-v The files are assumed to contain a raster image for devices like the Benson® Varian.

-f Use a filter that interprets the first character of each line as a standard FORTRAN carriage control character.

The remaining single-letter options have the following meanings:

-r Remove the file when spooling or printing (with the **-s** option) is complete.

-m Send mail on completion.

-h Suppress the printing of the burst page.

-s Use symbolic links. Usually files are copied to the spool directory.

The **-C** option has the following argument as a job classification to use on the burst page. For example,

```
lpr -C EECS foo.c
```

causes the system name (the name returned by *hostname*(1)) to be replaced on the burst page by EECS and the file **foo.c** to be printed.

The **-J** option has the following argument as the *job* name to print on the burst page. Normally, the first file's name is used.

The **-T** option uses the next argument as the *title* used by *pr*(1) instead of the file name.

To get multiple copies of output, use the **-#***num* option, where *num* is the number of copies for each file named. For example,

```
lpr -#3 foo.c bar.c more.c
```

would result in three copies of the file **foo.c**, followed by three copies of the file **bar.c**, etc. On the other hand,

```
cat foo.c bar.c more.c | lpr -#3
```

will give three copies of the concatenation of the files.

The **-i** option indents the output. If the next argument is numeric, it is used as the number of blanks to be printed before each line; otherwise, eight characters are printed.

The **-w** option takes *num* to be the page width for *pr*(1).

The **-s** option will use *symlink*(2B) to link data files rather than trying to copy them so that large files can be printed. This means the files should not be modified or removed until they have been printed.

FILES

/etc/passwd	personal identification
/etc/printcap	printer capabilities database
/usr/lib/lpd*	line printer daemons
/usr/spool/*	directories used for spooling
/usr/spool/*/cf*	daemon control files
/usr/spool/*/df*	data files specified in "cf" files
/usr/spool/*/tf*	temporary copies of "cf" files

SEE ALSO

lpq(1), lprm(1), symlink(2B), printcap(4).
lpc(1M), lpd(1M) in the *CLIX System Administrator's Reference Manual.*
pr(1) in the *UNIX System V User's Reference Manual.*

DIAGNOSTICS

lpr will object to printing binary files.

If a user other than the super-user prints a file and spooling is disabled, *lpr* will print a message saying so and will not put jobs in the queue.

If a connection to *lpd*(1M) on the local machine cannot be made, *lpr* will say that the daemon cannot be started.

Diagnostics may be printed in the daemon's log file regarding missing spool files by *lpd*(1M).

NOTES

lpr truncates files that are too large.

BUGS

Fonts for *troff* and *TeX* are on the host with the printer. It is not currently possible to use local font libraries.

NAME

lprm - remove jobs from the BSD line printer spooling queue

SYNOPSIS

lprm [**-P** *printer*] [-] [*job#* ...] [*user* ...]

DESCRIPTION

lprm removes jobs from a printer's spool queue. Since the spooling directory is protected from users, using *lprm* is normally the only method by which a user may remove a job.

Without arguments *lprm* will delete the currently-active job if it is owned by the user who invoked *lprm*.

If the - flag is specified, *lprm* will remove all jobs that a user owns. If the super-user employs this flag, the spool queue will be emptied entirely. The owner is determined by the user's login name and host name on the machine where the *lpr* command was invoked.

Specifying a *user's* name or a list of *user* names causes *lprm* to attempt to remove any jobs queued belonging to the *user*(s) This form of invoking *lprm* is useful only to the super-user.

A user may dequeue a job by specifying its *job* number. This number may be obtained from the *lpq*(1) program as follows:

```
$ lpq -l
1st: ken                      [job #013ucbarpa]
     (standard input)         100 bytes
$ lprm 13
```

lprm announces the names of any files it removes and is silent if no jobs in the queue match the request list.

lprm kills the *lpd*(1M) daemon if necessary before removing any spooling files. If a daemon is killed, a new one is automatically restarted on completion of file removals.

The **-P** option may be used to specify the queue associated with a specific *printer*. (Otherwise, the default printer or the value of the PRINTER variable in the environment is used).

FILES

/etc/printcap	printer characteristics file
/usr/spool/*	spooling directories
/usr/spool/*/lock	lock file used to obtain the pid of the current daemon and the job number of the currently-active job

SEE ALSO

lpr(1), lpq(1).
lpd(1M) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

"Permission denied" if the users try to remove files other than their own.

BUGS

Since race conditions are possible in the lock file update, the currently-active job may be incorrectly identified.

NAME

lptest - generate line printer ripple pattern

SYNOPSIS

lptest [*length* [*count*]]

DESCRIPTION

lptest writes the traditional "ripple test" pattern on standard output. In 96 lines, this pattern will print all 96 printable ASCII characters in each position. While originally created to test printers, it is quite useful for testing terminals, driving terminal ports for debugging purposes, or any other task where a quick supply of random data is needed.

The *length* argument specifies the output line length if the default length of 79 is inappropriate.

The *count* argument specifies the number of output lines to be generated if the default count of 200 is inappropriate.

NAME

ls - list contents of directory

SYNOPSIS

ls [**-RadCxmlnogrtucpFbqisfL**] [*name* ...]

DESCRIPTION

ls lists the directory contents for each directory argument; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format lists one entry per line. The **-C** and **-x** options enable multicolumn formats, and the **-m** option enables stream output format. To determine output formats for the **-C**, **-x**, and **-m** options, *ls* uses the environment variable COLUMNS to determine the number of character positions available on one output line. If this variable is not set, the *terminfo*(4) database is used to determine the number of columns based on the environment variable TERM. If this information cannot be obtained, 80 columns are assumed.

The following options are available:

-R Recursively list subdirectories encountered.

-a List all entries, including those that begin with a dot (.), which are normally not listed.

-d If an argument is a directory, list only its *name* (not its contents). This option is often used with **-l** to get the directory's status.

-C Print multicolumn output with entries sorted down the columns.

-x Print multicolumn output with entries sorted across rather than down the page.

-m Print in stream output format; files are listed across the page separated by commas.

-l List in long format giving mode, number of links, owner, group, size (in bytes), and time of last modification for each file (see below). If the file is a special file, the size field will contain the major and minor device numbers rather than a size. If the file is a symbolic link, the path name the link references is printed preceded by an arrow (—>).

-n Same as **-l** except that the owner's UID and group's GID numbers are printed rather than the associated character strings.

-o Same as **-l** except that the group is not printed.

-g Same as **-l** except that the owner is not printed.

-r Reverse the sort order to get reverse alphabetic or oldest first as appropriate.

-t Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See **-u** and **-c**.)

-u Use the last access time instead of modification time for sorting (with the **-t** option) or printing (with the **-l** option).

-c Use the last i-node modification time (such as when a file was created or a mode changed) for sorting (**-t**) or printing (**-l**).

-p Put a slash (/) after each file name if the file is a directory.

-F Put a slash (/) after each file name if the file is a directory, put an asterisk (*) after each file name if the file is executable, and put an at sign (@) after each file name if the file is a symbolic link.

-b Force nongraphics characters in file names to be printed in the octal **\ddd** notation.

-q Force nongraphics characters in file names to be printed as the character **?**.

-i For each file, print the i-node number in the first column of the report.

-s Give size in blocks (including indirect blocks) for each entry.

-f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **-l**, **-t**, **-s**, and **-r** and turns on **-a**; the order is the order in which entries appear in the directory.

-L If an argument is a symbolic link, print the information about the file or directory the link references rather than about the link itself.

The mode printed under the **-l** option consists of ten characters. The first character may be one of the following:

d	The entry is a directory.
b	The entry is a block special file.
c	The entry is a character special file.
l	The entry is a symbolic link.
p	The entry is a fifo (named pipe) special file.
—	The entry is an ordinary file.

The next nine characters are interpreted as sets of three bits each. The first set refers to the owner's permissions; the next set refers to permissions of others in the file's user group; and the last set refers to all others. Within each set, the three characters indicate (respectively) permission to read, write, and execute the file as a program. Execute permission for a directory is permission to search the directory for a specified file.

ls -l prints its output as follows:

```
−rwxrwxrwx   1 smith   dev     10876   May 16 9:42 part2
```

This horizontal configuration provides a lot of information. Reading from right to left, it is seen that the current directory holds one file, "part2". Next, the last time the file's contents were modified was 9:42 AM on May 16. The file is moderately sized, containing 10,876 characters, or bytes. The file owner, or the user, belongs to the group "dev", and the login name is "smith". The number (in this case "1") indicates the number of links to file "part2". Finally, the row of dashes and letters shows that user, group, and others have permission to read, write, and execute "part2".

The execute (**x**) symbol occupies the third position of the three-character sequence. A — in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

r	The file is readable.
w	The file is writable.
x	The file is executable.
—	The indicated permission is not granted.
l	Mandatory locking will occur during access. (The set-group-ID bit is on and the group execution bit is off.)
s	The set-user-ID or set-group-ID bit and the corresponding user or group execution bit are on.
S	Undefined bit-state. (The set-user-ID bit is on and the user execution bit is off).
t	The 1000 (octal) bit, or sticky bit (see *chmod*(1)), and the execution bit are on.
T	The 1000 bit is on and execution is off (undefined bit-state).

For user and group permissions, the third position is sometimes occupied by a character other than **x** or —. The **s**, referring to the state of the set-ID bit (the user's or the group's), may also occupy this position. For example, the ability to assume the same ID as the user during execution is used during login when the user begins as root but needs to assume the identity stated at login.

In the sequence of group permissions, **l** may occupy the third position. **l** refers to mandatory file and record locking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For others permissions, the third position may be occupied by **t** or **T**. These refer to the state of the sticky bit and execution permissions.

EXAMPLES

This example describes a file that the user can read, write, and execute and that group and others can read:

```
-rwxr--r--
```

This example describes a file that the user can read, write, and execute and that group and others can read and execute. This permission allows the user presently executing it to assume its user ID during execution:

```
-rwsr-xr-x
```

This example describes a file that only the user and group can read and write and that can be locked during access:

```
-rw-rwl---
```

This command will print the names of all files in the current directory, including those that begin with a dot (.), which normally do not print:

```
ls -a
```

This command will provide information such as all files (including non-printing ones (**a**)); the i-number, the memory address of the i-node associated with the file, printed in the left-hand column (**i**); and the size of the files (in blocks) printed in the column to the right of the i-numbers (**s**). The report is printed in the numeric version of the long list, printing the UID (instead of user name) and GID (instead of group name) numbers associated with the files.

```
ls -aisn
```

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	UIDs for **ls -l** and **ls -o**
/etc/group	GIDs for **ls -l** and **ls -g**
/usr/lib/terminfo/?/*	terminal information database

SEE ALSO

chmod(1), find(1).

NOTES

In a Remote File Sharing (RFS) environment, a user may not have the permissions that the output of the **ls -l** command implies. For more information, see the "Mapping Remote Users" section in Chapter 10 of the *UNIX System V System Administrator's Guide*.

BUGS

Unprintable characters in file names may confuse the columnar output options.

NAME

machid: clipper, ns32000, vax - get processor type truth value

SYNOPSIS

clipper
ns32000
pdp11
u3b
u3b2
u3b5
vax

DESCRIPTION

The following commands return a true value (exit code of 0) if the processor matches the command as follows:

clipper	CLIPPER-based system.
ns32000	NSC32000-based system.
pdp11	PDP-11/45 or PDP-11/70.
u3b	3B20 computer.
u3b2	3B2 computer.
u3b5	3B5 computer.
vax	VAX-11/750 or VAX-11/780.

These commands are often used within makefiles (see *make*(1)) and shell procedures (see *sh*(1)) to increase portability.

The commands that do not apply return a false (nonzero) value.

SEE ALSO

test(1).
sh(1), true(1), make(1) in the *UNIX System V User's Reference Manual.*

NAME

mailq - display a listing of the mail queue used by *sendmail*(1M)

SYNOPSIS

mailq [-**v**]

DESCRIPTION

mailq prints a listing of the mail queue used by *sendmail*(1M). For each message in the queue, *mailq* displays the queue ID, the message size, the date when the message entered the queue, the sender, and recipients. If the **-v** option is specified, the message priority is also displayed.

SEE ALSO

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

NAME

mailstats - display mail statistics

SYNOPSIS

mailstats [-f *file*]

DESCRIPTION

For each message received by or sent from the local machine, *sendmail*(1M) logs statistics, including the mailer name and the number of bytes in the message. *mailstats* reads this information and displays each mailer name, the number of messages sent or received through that mailer, and the total number of kilobytes sent or received through that mailer.

The **-f** option directs *mailstats* to read statistics from *file*. If *sendmail*(1M) is logging statistics to a file other than the default, **/usr/lib/sendmail.st**, the -f option must be specified for *mailstats* to read the statistics.

Invoking *mailstats* displays a list similar to the following:

```
Statistics from Thu Sep 6 14:30:43 1990

M   mailer_name   msgsfr   bytes_from   msgsto   bytes_to
0   local             99          99K       60        54K
2   ether              9           9K        2         2K
3   uucp              22          30K        1         1K
6   community         16          18K       41        39K
```

SEE ALSO

sendmail(1M) in the *CLIX Programmer's & User's Reference Manual.*

NAME

man - print entries in this manual

SYNOPSIS

man [*option* ...] [*section*] *title* ...

DESCRIPTION

man locates and prints the entry of this manual named *title* in the specified *section*. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) The *title* is entered in lowercase. The *section* number may not have a letter suffix. If no *section* is specified, the whole manual is searched for *title* and all occurrences of it are printed. *Options* and their meanings are as follows:

-T*term* Print the entry as appropriate for terminal type *term*. For a list of recognized values of *term*, type **help term2**.

-w Print on the standard output only the path names of the entries relative to **/usr/ip32/sysvdoc/catman** or to the current directory for the **-d** option.

-d Instead of **/usr/ip32/sysvdoc/catman**, search the current directory; requires the full file name (such as **cu.1c** rather than just **cu**).

-c Causes *man* to invoke *col*(1); note that *col*(1) is invoked automatically by *man* unless *term* is **300**, **300s**, **450**, **37**, **4000a**, **382**, **4014**, **tek**, **1620**, or **X**.

man examines the environment variable TERM (see *environ*(5)) and attempts to select options that adapt the output to the terminal being used. The -T*term* option overrides the value of TERM; in particular, the user should use **-Tlp** when sending the output of *man* to a line printer.

Section may be changed before each *title*.

The following example would reproduce this entry (*man*(1)) and any other entries named *man* that may exist in other sections of the manual on the terminal.

```
man man
```

FILES

/usr/ip32/sysvdoc/catman/?_man/man[1-8]/*

SEE ALSO

term(5) in the *UNIX System V Programmer's Reference Manual.*

CAVEATS

The *man* command prints manual entries that were formatted by *nroff* when the SYSVDOC product was installed. Entries are originally formatted with terminal type 37 and are printed using the correct terminal filters as derived from the -T*term* and TERM settings. Typesetting or other nonstandard printing of manual entries requires installation of the UNIX System V Documenter's Workbench^{TM}.

NAME

merge - three-way file merge

SYNOPSIS

merge [**-p**] *file1 file2 file3*

DESCRIPTION

merge incorporates all changes that lead from *file2* to *file3* in *file1*. The result goes to standard output if **-p** is present, or to *file1* otherwise. *merge* is useful for combining separate changes to an original. Suppose *file2* is the original and both *file1* and *file3* are modifications of *file2*. Then, *merge* combines both changes.

An overlap occurs if both *file1* and *file3* have changes in a common segment of lines. *merge* prints how many overlaps occurred and includes both alternatives in the result. The alternatives are delimited as follows:

```
<<<<<<< file1
lines in file1
=======
lines in file3
>>>>>>> file3
```

If overlaps occur, the user should edit the result and delete one of the alternatives.

SEE ALSO

rcsmerge(1), co(1).
diff3(1), diff(1) in the *UNIX System V User's Reference Manual.*

IDENTIFICATION

Author: Walter F. Tichy,
Purdue University, West Lafayette, IN 47907.
Copyright © 1982 by Walter F. Tichy.

NAME

mkshlib - create a shared library

SYNOPSIS

mkshlib -s *specfil* **-t** *target* [**-h** *host*] [**-n**] [**-L** *dir* ...] [**-q**]

DESCRIPTION

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

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

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

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

The *mkshlib* command invokes other tools such as the archiver, *ar*(1), the assembler, *as*(1), and the link editor, *ld*(1). Tools are invoked through the use of *execvp* (see *exec*(2)), which searches directories in the user's PATH. Also, prefixes to *mkshlib* are parsed in the same manner as prefixes to the *cc*(1) command, and invoked tools receive the prefix where appropriate. For example, **pfxmkshlib** will invoke **pfxld**.

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

-s *specfil* Specifies the shared library specification file, *specfil*. This file contains the information necessary to build a shared library.

-t *target* Specifies the output file name of the target shared library being created. It is assumed that this file will be installed on the target machine at the location given in the specification file (see the **#target** directive below). If the **-n** option is used, then a new target shared library will not be generated.

-h *host* Specifies the output file name of the host shared library being created. If this option is not given, the host shared library will not be produced.

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

-L *dir* ... Change the algorithm of searching for the host shared libraries specified with the **#objects noload** directive to look in *dir* before looking in the default directories. The **-L** option can be specified multiple times on the command line. In that case, the directories given with the **-L** options are searched in the order given on the command line before the default directories.

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

-v Print out the command line arguments of the programs invoked for the user.

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

All directives that can be followed by multiline specifications are valid until the next directive or the end of the file.

#address *sectname address*
Specifies the start address, *address*, of section *sectname* for the target. This directive typically is used to specify the start addresses of the **.text** and **.data** sections. One **#address** per section name is valid. An **#address** directive must be given exactly once for the **.text** section and once for the **.data** section. See the table in the section "The Building Process" in the "Shared Libraries" chapter of the *UNIX System V Programmer's Guide* for standard addresses.

#target *path-name*
Specifies the absolute path name, *path-name*, at which the target shared library will be installed on the target machine. The operating system uses this path name to locate the shared library when executing *a.out*(4) files that use this shared library. This directive must be specified exactly once per specification file.

#branch
Specifies the start of the branch table specifications. The lines following this directive are interpreted as branch table specification lines.

Branch table specification lines have the following format:

funcname <white space> *position*

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

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

This directive must be specified exactly once per shared library specification file.

#objects
The lines following this directive are interpreted as the list of input object files in the order they are to be loaded in the target. The list consists of each path name followed by a newline character. This list is also used to determine the input object files for the host shared library, but the order for the host is given by running the list through *lorder*(1) and *tsort*(1).

This directive must be specified exactly once per shared library specification file.

#objects noload
The **#objects noload** is followed by a list of host shared libraries. These libraries are searched in the order listed to resolve undefined symbols from the library being built. During the search, a nonshared version of a symbol found before a shared version of the symbol is an error.

Each name given is assumed to be a path name to a host or an argument of the form **-lX**, where **libX.a** is the name of a file in LIBDIR or LLIBDIR. This behavior is identical to that of *ld*(1), and the **-L** option can be used on the command line to specify other directories to locate these archives in.

Note that if a host shared library is specified using **#objects noload**, any *cc*(1) command that links to the shared library being built will need to specify that host also.

#hide linker [*]
This directive changes normally external symbols to static symbols, local to the library being created. A regular expression may be given (see *sh*(1) and *find*(1)). In this case, all external symbols matching the regular expression are hidden. The **#export** directive (see below) can be used to counter this effect for specified symbols.

The optional * is equivalent to

```
#hide linker
   *
```

and coverts all external symbols to static symbols.

All symbols specified in **#init** and **#branch** directives are assumed to be external symbols and cannot be changed to static symbols using the **#hide** directive.

#export linker [*]
Symbols given in the **#export** directive are external symbols (global among files) that, because of a regular expression in a **#hide** directive, would otherwise have been made static. For example,

```
#hide linker *
#export linker
   one
   two
```

tags all symbols except *one*, *two*, and those used in **#branch** and **#init** entries as static.

#init *object*
Specifies that the object file, *object*, requires initialization code. The lines following this directive are interpreted as initialization specification lines.

Initialization specification lines have the following format:

ptr <white space> *import*

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

```
ptr = &import;
```

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

#ident *string*
Specifies a string, *string*, to be included in the .comment section of the target shared library.

Specifies a comment. All information on the line beginning with **##** is ignored.

FILES

$TEMPDIR/*	temporary files TEMPDIR is usually **/usr/tmp** but can be redefined by setting the environment variable TMPDIR (see *tempnam*() in *tmpnam*(3S)).
$LIBDIR	usually **/lib**

$LLIBDIR	usually **/usr/lib**

SEE ALSO

as(1), cc(1), ld(1), a.out(4).
ar(1), chkshlib(1), lorder(1), tsort(1), ar(4) in the *UNIX System V Programmer's Reference Manual.*
"Shared Libraries" chapter in the *UNIX System V Programmer's Guide.*

CAVEATS

The **-n** option cannot be used with the **#objects noload** directive.

If *mkshlib* is asked to create a host library and a host when that name exists, *mkshlib* will update the host using *ar*(1) **-ru**. This means that the host should always be removed before rebuilding when an object file previously included in the library is removed or renamed.

If the address specified with the **#address** directive is outside user space, the library build may return successfully, but it might not work when it is used.

NAME

monparam - CRM utility for monitoring system parameters

SYNOPSIS

/usr/ip32/crm/monparam [**-I** *interval*] [**-i** *input-file*] [**-o** *output-file*]

DESCRIPTION

monparam monitors the parameters of a running system. For example, *monparam* can show whether the user is running out of a resource. This monitor is a complement to the configurable UNIX utility. (Configurable UNIX allows the user to change system parameters.)

The following options are available:

-I *interval*	Specify how frequently the monitor samples and displays information. *Interval* is the number of seconds. The default is 2.
-i *input-file*	Read the data from *input-file* each interval. *Input-file* must have been created as an *output-file* using the **-o** option. A - for *input-file* reads input from **stdin**.
-o *output-file*	Direct output to *output-file*. A - for *output-file* directs output to **stdout**.

The following describes each *monparam* field:

Sample time	Displays how frequently (in seconds) the monitor gathers and displays information. The default setting is two seconds. This time interval can be changed by pressing the up arrow key (to increment) and the down arrow key (to decrement).
Name	Displays the name of the parameter being monitored.
Current	Displays the current value of the parameter being monitored.
Max	Displays the maximum value of the parameter since the system was booted last.
Configured	Displays the value specified for the parameter under Configurable UNIX.

A parameter line will be highlighted when the maximum value is 90 percent or more of the configured value.

SEE ALSO

crm(1).

WARNINGS

Sending raw data to a file can create a very large file.

NAME

monproc - CRM utility for monitoring a process

SYNOPSIS

/usr/ip32/crm/monproc [**-w**] [**-I** *interval*] [**-o** *output-file*] *input-option*

DESCRIPTION

monproc monitors CPU use, status, priority, hard and soft fault rates, and current PC (program counter) for a process.

The following options are available:

-o *output-file*	Direct output to *output-file*. A - for *output-file* directs output to **stdout**.
-I *interval*	Specify how frequently the monitor samples and displays information. *Interval* is the number of seconds. The default is 2.
-w	Execute *monproc* in graphics-based format.

In graphics, to expand the list of processes being monitored, stretch the length of the window with the standard modify icon. To receive a description of each category represented in the monitor bar graphs, select the question mark (?) icon from the window icon box. A help window will appear.

In graphics, to change the colors of the bar graphs in *monproc*, select the color palette icon from the window icon box. A small Color menu will appear. The foreground color is displayed when the menu first appears. Clicking the mouse button moves to the next color. Exit and save the changes by selecting the delete icon in the Colors window. These colors are saved for the current monitoring session only.

The following *input-options* are available:

-i *input-file*	Read the data from *input-file* each interval. *Input-file* must have been created as an *output-file* using the **-o** option. A - for *input-file* reads input from **stdin**.
-p *pid*	Specify the ID number of the process to monitor (PID). The user can key in **ps -e** at the system prompt to determine the PID of a process already running.
-n *process-name*	Specify the name of the process to monitor. The user can key in **ps -e** at the system prompt to determine the name of a process already running.
-e *command* [*arg* ...]	Allow the user to run, provide arguments for, and monitor a program.

A brief explanation of the *monproc* fields follows.

CPU user time

CPU system time	Displays the amount of CPU time used by the process (user) and the system since the beginning of the monitoring session.
Status	Displays the process activity (such as SLEEP or STOP) when the CPU examines it.
Priority	Displays the priority assigned by the system to the process being monitored.
Username	Displays the user name that is running the process being monitored.
Hard fault rate Soft fault rate	Displays the number of hard and soft faults that occurred per second during the sample interval.
Physical Memory Virtual Memory	Displays the amount of physical and virtual memory the system assigns to the process being monitored.
Elapsed time	Displays how long the process has been running.
PC	Displays the address where the program counter was located the last time the monitor polled it. If the program was compiled to include debugger symbols (such as to be used by Intergraph's *dbg*(1)), the monitor can read those symbols and provide more logical values in this field. For instance, the PC might display a more logical address such as **sub1 + 10**, where **sub1** is the name of a procedure in the program and **10** is the number of bytes offset into **sub1**.

In graphics-based format, the first bar shows activity for the last sample period. The second bar shows average activity for the last 10 sample periods.

You will notice two separate color bar graphs when you execute the graphics-based process monitor. The first bar shows activity for the last sample period; the second bar shows average activity for the last 10 sample periods.

EXAMPLES

The following is an example of a *monproc* session:

```
Process is xns_listen  Thu May 25  11:14:53 1990

CPU user time 00 00:00:00.41  CPU system time 00 00:00:06.31
Status: STOP    Priority: 14    Username: root
Hard fault rate 00/sec Soft fault rate 00/sec
Physical Memory  196 k          Virtual Memory  532 k
Elapsed time:  00 00:52:16:00
PC:  0000efe6 /current
PC:  0000efe6 /last
```

SEE ALSO

crm(1).

WARNINGS

Sending raw data to a file can create a very large file.

NAME

monregion - CRM utility for monitoring memory regions

SYNOPSIS

/usr/ip32/crm/monregion [**-w**] [**-o** *output-file*] *input-option*

DESCRIPTION

monregion monitors the memory regions used by a specified process.

The following options are available:

-o *output-file*	Direct output to *output-file*. A - for *output-file* directs output to standard output.
-w	Execute *monregion* in graphics-based format.

In graphics, to expand the list of processes being monitored, stretch the length of the window with the standard modify icon. To receive a description of each category represented in the monitor bar graphs, select the question mark (?) icon from the window icon box. A help window will appear.

The following *input-options* are available:

-i *input-file*	Read the data from *input-file*. *Input-file* must have been created as an *output-file* using the **-o** option. A - for *input-file* reads input from **stdin**.
-p *pid*	Specify the ID number of the process to monitor (PID). The user can key in **ps -e** at the system prompt to determine the PID of a process already running.
-n *process-name*	Specify the name of the process to monitor. The user can key in **ps -e** at the system prompt to determine the name of a process already running.
-e *command* [*arg* ...]	Allow the user to run, provide arguments for, and monitor a program.

A brief explanation of the *monregion* fields follows.

TEXT DATA STACK	Displays three memory regions that are monitored for any process.
SHARED PRIVATE	Displays whether the region can be shared with other processes or is local to this process.
RDONLY RD/WRT	Displays whether the process has write access to the region or the region is read-only.
GROWDOWN	Displays whether or not the stack region is monitored according to stack region structure. Since a stack region is

structured to begin at high addresses and decrease to low addresses, the region is monitored from high to low. Pages added to a stack region are added at the lowest virtual address rather than the highest. Thus, when a stack region grows, it grows downward.

size
: Displays the number of virtual pages in the region.

valid
: Displays the number of physical pages mapped to virtual pages in the region. The rows of numbers, asterisks, ampersands, and other alphanumeric characters below the names of the memory regions provide information about the memory pages as follows:

  The alphanumeric characters (00000000, 00400000, etc) directly below the memory regions specify the starting address (in the region) for that line in the monitor. Since the activity of the entire region cannot be displayed on a single line in the monitor, the monitor breaks the region into several parts for display purposes.

  The asterisks (*) represent a physical page of memory mapped to that region. Every blank space between the * represents a page of virtual memory without a physical page mapped to it.

  The ampersand (&) indicates the memory page that the PC is on.

  The "L" indicates a physical page of memory that is locked to a process. A locked page cannot be taken from that process. For example, if the I/O system will need a page for an I/O request, the system will lock a page in memory until the process is finished with that page.

  The vertical bar (|) represents the end of the section of the memory region shown on that line.

EXAMPLES

The following is an example of a *monregion* session:

```
TEXT    SHARED   RDONLY      size:35    valid:23
00000000: *****  ***** **&*   *******|
DATA    PRIVATE  RD/WRT      size:45    valid:16
00400000: ******  **L       *      ********|
00440000:  ***** *  *  *  *** *|
00480000:    *  ***  ******* * *  ***   *|
STACK   PRIVATE RD/WRT GROWDOWN size:2      valid:1
bffffff: *|
```

SEE ALSO

crm(1).

WARNINGS

Sending raw data to a file can create a very large file.

NAME

mt - magnetic tape manipulation program

SYNOPSIS

mt [**-f** *tape-name*] *command* [*count*]

DESCRIPTION

mt is used to give commands to a magnetic tape drive. If a tape name is not specified with the **-f** option, the environment variable TAPE is used. If TAPE does not exist, *mt* uses the device **/dev/rmt/0mn**. By default, *mt* performs the requested operation once. Operations may be performed *count* times by specifying *count*.

The available *commands* are listed below.

eof, weof	Write *count* end-of-file marks at the current position on the tape.
fsf	Forward space *count* files.
fsr	Forward space *count* records.
bsf	Backward space *count* files.
bsr	Backward space *count* records.
rewind	Rewind the tape (*count* is ignored).
offline, rewoff	Rewind the tape and place the tape unit offline (*count* is ignored).
fseot	Forward space to end of recorded media (*count* is ignored).
erase	Erase the entire tape (*count* is ignored).
retension	Retension the tape (*count* is ignored).
density	Set recording density to *count*.
status	Print status information about the tape unit (*count* is ignored).

FILES

/dev/rmt/*
/dev/rmt/0mn

SEE ALSO

tc(7S) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

mt returns a 0 exit status when the *command* was successful, 1 if the *command* was unrecognized, and 2 if the *command* failed.

NAME

ncp - DNP network control program

SYNOPSIS

ncp [*command*]

DESCRIPTION

ncp (Network Control Program) provides the Digital Network Protocol (DNP) a set of interactive commands to configure, control, monitor, and test a Digital Network Architecture (DNA) network to ensure its effective operation.

Invoking *ncp* without *command* causes *ncp* to enter interactive mode and display the **NCP>** prompt on the next line. *ncp* commands are not case-sensitive except when *user-id* or *password* is specified. Lowercase letters are automatically converted to uppercase internally.

An *ncp command* consists of a command keyword, an entity, and one or more entity options that qualifies the command by supplying additional information. Keywords may be abbreviated as long as they are still unique and are a minimum of three characters.

The **clear**, **define**, **disconnect**, **set**, and **zero** commands require super-user privileges. All users can use the **show**, **loop**, and **tell** commands.

The **set**, **clear**, and **show** commands deal with information in the volatile database. **define** commands affect the permanent database.

General Definitions

The following are some general definitions of the terminology used.

end node	An end node has a single circuit connecting it to the rest of the network. An end node can send packets to any other DNA node and receive packets addressed to itself from other Phase IV nodes. An end node is a nonrouting node. It cannot route packets.
local node	The local node where the user is physically located.
remote node	Remote nodes are all other nodes in relation to the local node.
executor node	The executor node is where *ncp* commands are executed. The executor node is usually the local node. However, through the *ncp* command **set executor**, any remote node can be designated as the executor node; thus, any *ncp* commands issued on the local node are executed on the remote node.
reachable or active node	A reachable or active node is one that is available for connection requests. A node is unreachable or

	inactive when it is powered down or unavailable for connection requests.
adjacent node	An adjacent node is connected to the local node by a single physical line. All nodes on a single Ethernet line are considered adjacent.
node state	The node state is the operational state of the local node on the network. This state can be controlled and is used to restrict the operation of the node or to shut it down. When the state is set to off, the node is unreachable. When the state is set to on, the node is active or reachable.
circuits	Circuits are logical communications data paths between nodes. They operate over physical lines. Ethernet circuits enable multiaccess connection between a number of nodes on the same physical medium. Each node is considered adjacent to every other node on the circuit and is equally accessible. Each node is identified by an Ethernet address.
lines	Lines are physical data paths between nodes and are the lowest-level communications path. The Ethernet line physically connects the different nodes on the local network.
counters	Counters are performance variables that track various events in a network. Information obtained from counters may be useful in measuring the performance and throughput for a given circuit. There are counters tracking performance on nodes, circuits, and lines. Node counters are available only after a connection has been attempted between the executor and the specified node. If no logical link has been established between the executor and the specified node, the **show node** *node-id* **counters** command returns with the message "NO INFORMATION AVAILABLE."

Node Identification

Many of the commands allow a *node-id* to be specified. *Node-id* can either be a *node-name* or an *node-address*.

node-name	Specifies a node name. A node name can have up to six alphanumeric characters. At least one character of the node name must be a letter. A node name is not case-sensitive. Lowercase letters are converted to uppercase.

The address and name for the local and remote nodes are stored in the configuration database. The configuration database for the local node must contain information about the local node and the remote nodes.

node-address Specifies a node address in the following form:

[*area-number*.]*node-number*

area-number is a group of nodes in the network that can run independently as a subnetwork. Each area has a unique number in a network. The area number must be an integer in the range of 1-63. The area number defaults to the local area if none is provided.

node-number must be an integer in the range of 1-1023. Node numbers must be unique to the specific network area.

Commands

ncp includes the following *commands*. **show** commands are described in separate subsections.

clear executor node
Reset the executor to the local node. Clear the default executor node designation previously specified through the **set executor node** command.

clear node *node-id* **all**
Identify the remote node whose parameters will be removed from the volatile database.

clear node *node-id* **all** should be used with caution, especially if links are open to the node being cleared. Before this command is invoked, the *ncp* **set executor state off** or **disconnect known links** command should be used.

define executor all
copy the contents of the volatile database to the permanent database.

define known nodes all
Copy, for each of the known nodes, the contents of each volatile database record to the permanent database.

define *node-id* **all**
Copy, for the specified *node-id*, the contents of the volatile database to the permanent database. *Node-id* is a node name or address.

disconnect link *link*
Disconnect a logical link *link*. Use the *ncp* command, **show known links**, to determine the link number to use for this command.

The **disconnect link** commands are used primarily to recover from network failure. The **disconnect link** commands cause applications using the specified links to fail and should be used with caution.

disconnect known links
: Disconnect all logical links. This command should be used with caution. See the **disconnect link** command.

exit
: Exit *ncp* from the interactive session. Alternately, a <CONTROL>-D can be used.

loop executor [count *count* **] [length** *length* **]**
[with {zeros|ones|mixed}] [user *user-id* **] password** *password*

Test the logical links within a single node. This test is performed by looping messages to the loopback mirror on the local node. The options have the following meanings:

count *count* Specify the number of times the command will be repeated. The default is 1 and the maximum is 65535.

count *length* Specify the number of bytes in the loop message. The default value is 40 and the maximum value is 245.

with {zeros|ones|mixed}
Specify the test data. The default is **mixed.**

user *user-id* Specify the user name to be used for access control information in connecting to the remote mirror.

password *password*
Specify the password that corresponds to *user-id.*

loop node *node-id* **[count** *count* **] [length** *length* **]**
[with {zeros|ones|mixed}] user *user-id* **password** *password*

Test the logical links to a remote node. The options have the following meanings:

node *node-id* Identify the node name or address.

count *count* Specify the number of times the command will be repeated. The default is 1 and the maximum is 65535.

length *length* Specify the number of bytes in the loop message. The default value is 40 and maximum is value 245 a CommUnity or CLIX remote node or 1458 for another DECnet remote node.

with {zeros|ones|mixed}
Specify the test data. The default is **mixed.**

user *user-id* Specify the user name to be used for access control information in connecting to the remote mirror.

password *password*
Specify the password that corresponds to *user-id.*

set executor address *node-address*
Specify a new node address for the executor node. This command can be issued only if the state of the executor is off. It is not possible to issue *ncp* commands to a remote executor node when the state of that node is off. Therefore, it is essential, when this command is issued, that the executor node be the local node. *node-address* specifies the node address for the local (executor) node.

set executor all
Copy the contents of the permanent database to the volatile database for the executor node.

set executor delay factor *factor*
Set the delay factor. *Factor* is in the 0-255 range. This value is multiplied by one sixteenth of the estimated round trip delay time to determine the appropriate value for the time to retransmit certain Network Server Protocol (NSP) messages. The default value delay factor is 80.

set executor delay weight *weight*
Set the delay weight. *Weight* is in the 0-255 range. NSP estimates the current delay in round trip transmission to a node with which it is communicating. *Weight* is used to calculate a new value of the estimated round trip delay. The default delay weight is 5.

set executor inactivity timer *timer*
Set the inactivity timer interval. A logical link is inactive when no data is transmitted in either direction for a given interval of time. The inactivity timer regulates the frequency with which CLIX tests the viability of an inactive link. The inactivity timer parameter is used to specify the maximum duration of inactivity before the local CLIX node tests the viability of the link. When the timer expires, CLIX generates artificial traffic to test the link. The default inactivity timer value is 10.

set executor name *node-name*
Specify *node-name* as the executor node name.

set executor node *node-id* [**userid** *user-id* **password** *password*]
Specify the local or remote node as the executor for all subsequent *ncp* commands.

node *node-id*	Identify the local or remote node name or address.
userid *user-id*	Identify the user-id on the remote system for the connection.
password *password*	Specify the password associated with *user-id*.

If a user is not specified, the default account on the target node is used. The **clear executor node** command resets the executor to the local node.

set executor retransmit factor *factor*

Set the retransmit factor. *Factor* specifies the number of times a packet may be retransmitted before a link is declared broken. The value of the retransmit factor regulates the number of times the NSP layer reattempts a transmission when its retransmission timer expires for a logical link. A number in the 0-65535 range should be used for this value. The default retransmit factor is 10.

set executor segment buffer size *size*

Specify, in bytes, the maximum *size* of transmit buffers for the executor node. *Size* is in the 255-1458 range. The default value is 1458.

When a logical link is established between two nodes, the nodes exchange their segment buffer sizes. The smaller of the two sizes is used as the negotiated segment buffer size for the link. Two end nodes may agree on a segment buffer size but have an intermediate router with a smaller segment buffer size (such as a router handling both DDCMP and Ethernet circuits). The end nodes can communicate unless one of them transmits a packet that exceeds the router's segment buffer size. When this happens, the router truncates the packet and the logical link is broken.

set executor state {on|off}

Turn the state of the node off and on. When the node state is on, the node is reachable from other network nodes; that is, new logical links to that node can be created. When the node state is off, the node is unreachable.

on	Allow creation of new logical links. **on** is the normal operational state of a node.
off	Prevent creation of new logical links, terminate existing links, and shut down the node.

Setting the state off terminates any active links to the node and could result in loss of data.

set known nodes all

Copy the contents of the permanent database to the volatile database for all nodes in the database.

set node *node-id* **address** *node-address* **name** *node-name*

Specify remote node names and address when building the executor's volatile configuration database.

node-id	Identifies the local node and can be a name or an address.
node-address	Specifies the address of the node to be included in the configuration database.

node-name	Specifies the name of the node to include in the configuration database. Only one name can be assigned to a node address. Duplicate node names are not permitted.

set node *node-id* **all**
Copy the contents of the permanent database to the volatile database for a single node *node-id*.

tell *node-id command*
Identify the executor for a particular *ncp* command. The *node-id* is set for only one *ncp* command.

zero executor counters
Reset all counters to zero on the executor node.

zero known circuit counters
Reset circuit counters to zero for all known circuits.

zero known line counters
Reset line counters to zero for all known lines.

zero known nodes counters
Reset node counters to zero for all known nodes.

zero node *node-id* **counters**
Reset node counters to zero for *node-id*. *Node-id* is the name or address of a local or remote node.

Show Characteristics Commands

show executor characteristics
show node *node-id* **characteristics**
show active nodes characteristics
show known nodes characteristics
Display static node information for the executor, a specified *node-id*, all active nodes, or all known nodes. The following information is included in the display:

Identification
Management Version
NSP Version
Maximum Links
Delay Factor
Delay Weight
Inactivity timer
Retransmit factor
Routing Version
Type
Maximum Address
Max Broadcast Nonrouters
Segment buffer size

Show Counters Commands

show executor counters
show node *node-id* **counters**
show active nodes counters
show known nodes counters

Display information about the user traffic between the executor and the specified node. The following information is included in the display:

Seconds since last zeroed
User data bytes received
User data bytes sent
User data messages received
User data messages sent
Connects received
Connects sent
Response timeouts
Received connect resource errors
Packet format errors

Show Status Commands

show executor status

Display executor status information. This consists of the node name and address, state, and Ethernet physical address.

show node *node-id* **status**
show active nodes status

Display the status of *node-id* or all nodes. *Node-id* is is a node name or address. The display consists of the following information:

Node address and name
Routing state
Number of active logical links associated with the node
Delay timer to set the retransmission
Node type

Show Summary Commands

show executor summary
show node *node-id* **summary**
show active nodes summary
show known nodes summary

Display summary information. The display includes the following information:

Node address and name
Routing state
Number of active logical links associated with the node
Delay timer to set the retransmission
Node type

Show Circuit Commands

show known circuit characteristics

Display circuit characteristics. The following information is included in the display:

State
Designated router
HELLO timer
Type
Adjacent node
Listen timer

show known circuit counters

Display circuit counters. The following information is included in the display:

Seconds since last zeroed
Data blocks sent
Data blocks received
Bytes sent
Bytes received

show known circuit status

Display circuit status. The following information is included in the display:

Circuit ID
Circuit current state
Address and name of adjacent nodes on that circuit
Adjacent node ID
Block size

show known circuit summary

Display circuit summary. The following information is included in the display:

Circuit ID
Circuit current state
Address and name of adjacent nodes on that circuit
Adjacent Node ID
Block size

Show Line Commands

show known line characteristics

Display line characteristics. The display includes the line's protocol and hardware address.

show known line counters

Display line counters. The display includes the following information:

Seconds since last zeroed
Data blocks received

Data blocks sent
Blocks sent, multiple collisions
Collision-detect check failure
System buffer unavailable

show known line status
Display line status. The display includes the line and state.

show known line summary
Display line summary. The display includes the line and state.

Show Links Command

show known links
Display information for all known links connected to the local node.

SEE ALSO

Digital Network Protocol (DNP) Network Manager's Guide.

NAME

netaddr - display network address

SYNOPSIS

netaddr

DESCRIPTION

netaddr returns the network address for the node executing this command. Depending on which communication protocols are running, a subset of the following information is returned:

> *LAN-number.host-number*
> **Interface name "*xx*":**
> **Internet Address:** *nnn.nnn.nnn.nnn*
> **Subnet Mask:** *nnn.nnn.nnn.nnn*

SEE ALSO

"BSD Network Configuration Tutorial" in the *CLIX System Guide*.
Intergraph Network Core User's Guide.

NAME

netcp - DNP copy command

SYNOPSIS

netcp [**-ilnrtvxz**] *file1 file2*
netcp [**-ilnrtvxz**] *file ... directory*
netcp -r

DESCRIPTION

netcp copies files between hosts that support the Digital Network Architecture (DNA). This includes DECnet, CommUnity, and CLIX hosts on a network. File and specifications can be either simple file specifications of local files or the lengthy DNP remote file specifications.

The following options are available:

-i Set interactive mode. Prompt the user to confirm each file copy operation by entering one of the following responses:

Y or **y**	Copy the file and continue the interactive file copy mode.
N or **n**	Do not copy the file and continue the interactive file copy mode.
R or **r**	Copy the file and all the remaining files. This terminates the interactive file copy mode.
Q or **q**	Quit.

The interactive option is particularly useful in a selective transfer with wildcard specification.

-l Set logging mode. Print logging information on the standard output to indicate the start of data transfer for each file.

-n Set noisy mode. Print a message on the standard error stream indicating when there is an attempt to connect to *fal*(1M), the remote file transfer server. This often takes several seconds, and the message provides a way to monitor the operation.

-r Display release number. Specify the release and revision numbers of *netcp* and its components. If the release number switch is the sole argument to *netcp*, *netcp* prints the release information and terminates.

-t Display the total number of bytes and files transferred.

-v Set verbatim mode. Transfer (byte for byte) all input files without record format conversion and with no bytes lost, altered, or inserted. Output files are created with a record format appropriate to their byte-stream nature. On VMS, the output files always have VARIABLE RECORD format and NO RECORD attributes. When data is copied from one CLIX system to another CLIX system, the verbatim mode increases copying speed.

-x Submit the input files for execution on the remote system. These files are deleted after execution.

-z Set append mode. Append the input files to the destination files rather than overwriting them.

Options to *netcp* can be placed anywhere on the command line and in any order.

File-spec is a DNP file specification that can be specified in one of the following ways:

[*node-spec*["*username* [*password* [*account*]] "]::]*file-spec*

[**-u** *username*] [**-p** *password*] [**-a** *account*] [*node-spec*::]*file-spec*

Node-spec specifies a DECnet, CommUnity, or CLIX host name or address. The optional information enclosed in double quotation marks or specified with the **-u**, **-p**, or **-a** option is regarded as the access information. The remote system uses this infomation to determine accessibility on the remote host. The final portion of the syntax is the file specification on the remote host. The keywords are defined as follows:

node-space Specifies a Digital Network Architecture (DNA) host name or address. For example, DECnet, CommUnity, and CLIX hosts support DNA. The name or address is defined as follows:

node-name Specifies a host name. *Node-name* can be up to six characters long.

[*area-number*.]*node-number*
Specifies an address. The optional *area-number* is an integer in the range of 1-63 that specifies the network area of the host. *Node-number* is an integer in the range of 1-1023 that is unique in the network area. If the remote *node-number* is located in the same local network area, *area-number* need not be specified.

username
-u *username* Identifies the user on the remote system in whose name the access will be performed. The **NET_USER** environment variable, if defined, is used if no *username* is specified on the command line.

password
-p *password* Specifies *password* for *username*. A null *password* can be specified with "".

account
-a *account* Indicates the party to be billed for network access time. This option is used by some DECnet systems. It is not valid for CLIX systems. The **NET_ACCOUNT** environment

variable, if defined, is used if no *account* is specified on the command line. A null *account* can be specified with "".

file-spec Specifies a file conforming to naming conventions on the remote host. UNIX-, VMS-, and MS-DOS-style file specifications are examples of some file-naming conventions. Copying files between hosts using different file-naming conventions may produce unexpected results. *File-spec* may be a wildcard specification.

The standard input device, such as the keyboard, can be used instead of the source input file by using a -. A standard output device can be used instead of *file2* or *directory* by using a -.

When multiple source input *files* are specified, the target *directory* must be a remote or local directory or standard output. The output files retain much of their original names. The destination node may shorten some file names.

When remote files are copied to a target directory on CLIX, their names are converted, if necessary, to names that are suitable for use on the CLIX system. Files are stripped of version numbers, and if the files are from a non-case-sensitive system like VAX/VMS, they are converted to lowercase. For example, **SYS$SYSDEVICE:[LEE.PROJ1]MYFIL.RNO** becomes **myfil.rno** on CLIX. The names of files from another CLIX or case-sensitive system are unchanged.

SEE ALSO

fal(1M) in the *CLIX System Administator's Guide.*

CAVEATS

VMS file type VARIABLE FIXED CONTROL (VFC) is not supported.

NAME

netex - DNP remote file execution utility

SYNOPSIS

netex [**-ilnr**] *batch-file* ...
netex -r

DESCRIPTION

netex is a Digital Network Protocol (DNP) command that allows CLIX users to execute *batch-file*s on a remote system. *netex* locates the *batch-file* on the remote system and sends an access message to the remote system to submit that file to execute. After the *batch-file* executes, the submitted *batch-file* remains unchanged.

Batch-file is a standard DNP remote file specification as described in *netcp*(1). The following options are allowed:

-i Set interactive mode. Prompt the user to confirm each file copy operation by entering one of the following responses:

Y or **y**	Execute the file and continue the interactive file execution mode.
N or **n**	Do not execute the file and continue the interactive file execute mode.
R or **r**	Execute the file and all remaining files. This terminates the interactive file execution mode.
Q or **q**	Quit.

The interactive option is particularly useful in a selective execution with wildcard specification.

-l Set logging mode. Print logging information to the terminal to indicate the start of the operation.

-n Set noisy mode. Print a message on standard error indicating when there is an attempt to connect to *fal*(1M), the remote file transfer server. This often takes several seconds, and the message provides a way to monitor the operation.

-r Display the release and revision numbers of *netex* and its components. If the release number switch is the sole argument to *netex*, *netex* prints the release information and terminates.

SEE ALSO

netcp(1).
fal(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

netlpr - DNP command to print file(s) on remote printers

SYNOPSIS

netlpr [+**q** [*queue-spec*] [+**n** *node*] [+**u** *user*] [+**p** *password*]
[+**a** *account*] [-*option* ...] *file* ...

DESCRIPTION

netlpr is a Digital Network Protocol (DNP) utility that prints any accessible text file on any printer attached to a remote host supporting the Digital Network Architecture (DNA), including the local node. The arguments to *netlpr* are remote file specifications as described in *netcp*(1), optionally interspersed with the following options:

+**q** [*queue-spec*]
: Specify, in the form of a remote file specification, the printer on which to print all files specified up to the next +**q** option or the end of the command line. The form of *queue-spec* is as follows:

  [[*node-spec*["*access-info*"]::][*queue-name*]

  The keywords have the following meaning:

  node-spec
  : Specifies either a host name or address as described in *netcp*(1). If *node-spec* is not specified, *queue-name* refers to a local queue.

  "*access-info*"
  : Specifies optional access control information used to access *queue-name* on the remote host. The syntax of *access-info* is as follows:

    "[*user* [*password* [*account*]]]"

    User, *password*, and *account* are equivalent to the +**u**, +**p**, and +**a** options. The meaning, as applied to remote file specification, is described in *netcp*(1).

  queue-name
  : Specifies the remote queue to be used. The +**q** option is required to specify any printer other than the local default printer. If a *node-spec* is given and no *queue-name* is specified, output goes to the default printer on that *node-spec*.

+**n** *node*
: Specify the default node on which to search for subsequently named files. The current default access control information remains in effect.

+**u** *user*
: Specify the default user name to use when accessing files on remote nodes. The +**u**, +**p**, and +**a** options in *netlpr* have the same meaning as the -**u**, -**p**, and -**a** options in

netcp(1).

+p *password* Specify the default password to use when accessing files on remote nodes.

+a *account* Specify the default account to use when accessing files on remote nodes.

-option Treat any argument beginning with a - as an argument for the local print spooling program. The interpretation of such arguments is controlled by the *netlpr* configuration file described below.

netlpr works with a local spooling program (see *lp*(1), *lpr*(1), and *qpr*(1)) or a printer device driver. The interface to the local spooling program or printer must be defined in the **/usr/lib/netlpr.cf** file for *netlpr* to function.

For example, the **netlpr.cf** file for *lpr*(1) is as follows:

```
/usr/bin/lpr
PrCJTi1234
```

The format of the configuration file using a spooling program is as follows:

The first line contains the full path name of the spooling program.

The first character of the second line indicates the option to use when specifying a nondefault local printer. For example, if this character is P, *netlpr* invokes the local spooler with the following command:

lpr -P printer *file*

If this facility is not provided by the local spooling program, this character should be a space.

The second character of the second line indicates the option that specifies that a file will be deleted after printing. If this facility is not provided by the local spooling program, this character should be a space.

The third and succeeding characters of the second line represent the set of options to the spooler that take a separate argument, as in the following line:

lpr -o *value file*

rather than

lpr -o*value file*

The nondefault printer and delete options mentioned above may be members of this set. If no spooler options behave in this manner, a single space should be inserted in this field.

If no spooling program is available, **netlpr.cf** should contain a single line giving the name of the device in **/dev** to which printed output should be sent. The name given should not start with a **/**. For example, if the device is **/dev/lp, netlpr.cf** should contain only the following:

```
lp
```

netlpr may be used as a replacement for *lpr* and then appears to behave identically as long as the user prints only local files and does not use any options specific to *netlpr*. The only requirement in this case is that the local spooling program must remain present on the system and **netlpr.cf** must contain a valid description of its location and characteristics.

EXAMPLES

The following command prints "file1" and "file2" on the local default printer:

```
netlpr file1 file2
```

The following command prints "file1" on the local default printer and "file2" on the local printer named "printer."

```
netlpr file1 +q printer file2
```

The commands issued by the previous example (using the example **netlpr.cf** above) are as follows:

```
lpr file1
lpr -Pprinter file2
```

The following examples are equivalent:

```
netlpr nodea"joe montana"::file1
netlpr +n nodea +u joe +p montana file1
```

Both examples print "file1," resident on "nodea" (accessed with *user-name* "joe" and *password* "montana"), on the local default printer. They then delete the file after printing. *netlpr* copies the file to "/tmp" on the local node and then issues the following command to *lpr*(1):

```
lpr -r /tmp/file1
```

The following command prints "file1" from "nodea" on the local default printer as in the previous example and prints "file2" from "nodeb," also accessed using *user-name* "doug" and *password* "flutie" on the same printer:

```
netlpr nodea"doug flutie"::file1 +n nodeb file2
```

The following command prints "file1" and "file2," both from "nodea" and accessed using the same access control information, on the local default printer:

```
netlpr +n nodea +u dan +p marino file1 file2
```

The following command prints the local file "file1" on the default printer attached to node "vax":

```
netlpr +q vax:: file1
```

The following command prints "file1," resident on "vax2," on the default printer attached to "vax1." It does this by copying the file to "vax1" along with a message specifying that it will be printed and subsequently deleted when the copy is complete.

```
netlpr +q vax1:: vax2::file1
```

The following command prints the local "file1" on printer "lca0" attached to node "vax" (presumably a VMS node):

```
netlpr +q vax::lca0: file1
```

SEE ALSO

netcp(1), lpr(1), lp(1), qpr(1).

CAVEATS

There is a known problem with using any spooler program that does not copy the file to be printed to a saved area for the spooler. The spooler program selected must (by default) copy the file to be printed when it is being scheduled for printing. *lp*(1) does not copy the file by default, but *lpr*(1) and *qpr*(1) do copy the file by default. The **netlpr.cf** file indicates to *netlpr* the spooler to use.

NAME

netls - DNP command that lists the directory contents on a remote system

SYNOPSIS

netls [**-chlrstU1**] *name* ...
netls -r

DESCRIPTION

netls is a Digital Network Protocol (DNP) utility that lists the contents of directories. *name* is a remote file specification as defined in *netcp*(1). If *name* is a file specification, the files matching it are listed. The file specification must conform to the wildcard rules of the remote system. By default, the files are sorted alphabetically by name.

-c List files by the creation time instead of the last modification time. When used with the **-t** option, this option sorts files according to the creation time. This works only with -l.

-h Do not display headers that describe the directory that the following files belong to. (By default, they are displayed.)

-l List files in long format. The protection mode, owner, size in bytes, and last modification time are listed with the file name.

When the long output format is requested, the 12-character protection mode is printed showing the following four protection levels:

1. System, for the system user (a VMS concept).
2. Owner, for the file owner.
3. Group, for users in the owner's group.
4. World, for all other users.

Each set of three characters specifies the privileges for designated users to read, write, and execute a file.

The privileges are as follows:

r	file may be read
w	file may be written to
x	file may be executed
-	permission denied

-r Display the release and version level of *netls*.

-s Print the file size (in kilobytes) before the rest of each file's information.

-t Sort by time stamp (latest first) instead of by name. The default is the last modification time.

-U List the time of the last access instead of the modification time. When used with the -**t** option, sort files according to the time of last access. This works only with -l.

-1 Print only one file entry on each line. This is the default mode for long format or when the standard output is not a terminal.

When *netls* generates multicolumn output, it checks the environment variable **COLUMNS** for the number of columns that can be displayed on the standard output device. (The default is 80.) *netls* formats each line of the listing accordingly.

SEE ALSO

netcp(1).
fal(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

netmsg - send a message to console devices on the local XNS network

SYNOPSIS

/usr/ip32/inc/netmsg [-?] [-**y**] [-**n** *node*] [*message*]

DESCRIPTION

netmsg broadcasts *message* to the console devices of all machines on the Xerox Network Services (XNS) local area network (LAN) running the *xns_listener*(1M). If no *message* is supplied on the command line, the user is prompted to enter a one-line message that is broadcast upon confirmation.

-**n**	Sends *message* to the console on *node* rather than to all machines.
-**y**	Suppresses confirmation. If the -**n** and -**y** options are not present, *netmsg* will prompt for confirmation before transmitting *message* to the entire network.
-?	Displays a usage message.

If no options are specified, a usage message is displayed.

SEE ALSO

Intergraph Network Core User's Guide.

NAME

netmv - DNP command that moves or renames one or more files

SYNOPSIS

netmv [**-ilnr**] *source-filespec destination-filespec*
netmv [**-ilnr**] *file* ... *destination-filespec*
netmv -r

DESCRIPTION

netmv moves a file or a group of files using the Digital Network Protocol (DNP). When *netmv* moves a file, two file specifications are required: *source-filespec*, or a number of *files*, and the *destination-filespec*. A *source-filespec* is a valid remote file specification as described in *netcp*(1) that may contain wildcards. A *destination-filespec* may contain a directory name, a file name, or both. When multiple *files* are being moved, the destination must be a directory. In any case, *destination-filespec* may not contain a node specifier or Access Control Information (ACI) because they are assumed to be the same as *source-filespec*. The following options are available:

-i Set interactive mode. Prompt the user to confirm the operation before each input file is copied

1 **Y** or **y** **T** or **t**	Move the file and continue the interactive file move mode.
N or **n**	Do not move the file and continue the interactive file move mode.
R or **r**	Move the file and all remaining files. This terminates the interactive file move mode. The interactive option is particularly useful in a selective move with wildcard specification.

-l Set logging mode. Display an acknowledgement on the screen for each file moved when the move is successful.

-n Set noisy mode. Print a message on the standard error stream indicating when there is an attempt to connect to *fal*(1M), the remote file transfer server. This often takes several seconds, and the message provides a way to monitor the operation.

-r Display the release number. Specify the release and revision numbers of *netmv* and its components. If the release number switch is the sole argument to *netmv*, *netmv* prints the release information and terminates.

SEE ALSO

netcp(1).
fal(1M) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

When an error occurs during *netmv* command execution, an error message in the following form is displayed:

node::*file-spec* <error description>

The error message is displayed even if the logging option was not selected.

NAME

netrm - DNP command that removes files

SYNOPSIS

netrm [**-ilnr**] *filespec* ...
netrm -r

DESCRIPTION

netrm deletes specified *filespec*s using the Digital Network Protocol (DNP). Both the remote file specification and the methods of specifying access control information are as described in *netcp*(1).

-i Set interactive mode. Prompt the user to confirm each file deletion by entering one of the following responses:

Y or **y**	Delete the file and continue the interactive file deletion mode.
N or **n**	Do not delete the file and continue the interactive file deletion mode.
R or **r**	Delete the file and all remaining files. This terminates the interactive file deletion mode.
Q or **q**	Quit.

-l Set logging mode. Print an acknowledgement, following the deletion of a remote file, on the standard output terminal.

-n Set noisy mode. Print a message on the standard error stream indicating when there is an attempt to connect to *fal*(1M), the remote file transfer server. This often takes several seconds, and the message provides a way to monitor the operation.

-r Display the release number. Specify the release and revision numbers of *netrm* and its components. If the release number switch is the sole argument to *netrm*, *netrm* prints the release information and terminates.

SEE ALSO

netcp(1).
fal(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

newaliases - rebuild the database for the mail aliases file

SYNOPSIS

newaliases

DESCRIPTION

newaliases rebuilds the random access database for the *sendmail*(1M) aliases file, **/usr/lib/aliases**. It must be run each time **/usr/lib/aliases** is changed.

SEE ALSO

aliases(4).
sendmail(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

npmount, npumount - mount and unmount file system

SYNOPSIS

/usr/bin/npmount [-r] [-f *fstype*] *special directory*
/usr/bin/npmount [-r] -f **NFS** [,*options*] *resource directory*
/usr/bin/npmount [-r] [-c] -d *resource directory*
/usr/bin/npumount *directory*
/usr/bin/npumount -d *resource*

DESCRIPTION

npmount and *npumount* run *setuid*(2) to set the user ID (UID) to root, allowing a nonprivileged user to mount and unmount file systems, with certain access restrictions.

The following restrictions apply for *npmount* or *npumount*:

1. The user must have write permission on both the mount point and its parent directory. If not, the command will fail with an error message.
2. The user must have read permission on the disk partition being mounted. If not, the command will fail with an error message.
3. *npumount*(1M) requires that the user have write permission in the parent directory of the mount point.

npmount and *npumount* accept exactly the same arguments as *mount*(1M) and *umount*(1M), respectively. The only exception is that *npmount* will print a usage summary if executed with no arguments; *mount*(1M) will print a list of currently mounted file systems.

The following options are available:

-r	Indicates that *special* or *resource* is to be mounted read-only. If *special* or *resource* is write-protected, this flag must be used.
-d	Indicates that *resource* is a remote resource that is to be mounted on *directory* or unmounted. To mount a remote resource, Remote File Sharing (RFS) must be running and the resource must be advertised by a remote computer (see *rfstart*(1M) and *adv*(1M)).
-c	Disables RFS client caching of file system reads and writes on this *resource*.
-f *fstype*	Indicates that *fstype* is the file system type to be mounted. If this argument is omitted, it defaults to the root *fstype*. If *fstype* is Network File System (NFS), NFS options may be added after the *fstype* separated by commas. The available NFS options are as follows:

soft	Return an error if the server does not respond.
rsize=*n*	Set the read buffer size to *n* bytes.

wsize=*n*	Set the write buffer size to *n* bytes.
timeo=*n*	Set the initial NFS timeout to *n* tenths of a second.
retrans=*n*	Set the number of NFS retransmissions to *n*.
port=*n*	Set the server IP port number to *n*.

special Indicates the block special device to be mounted on *directory*. If *fstype* is NFS, *special* should have the form *host-name:/path-name*.

resource Indicates the remote resource name to be mounted on a directory.

directory Indicates the directory mount point for *special* or *resource*. (The directory must exist.)

SEE ALSO

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

NAME

odcd - set the current default directory used by optical disk commands

SYNOPSIS

odcd *odpathname*

DESCRIPTION

odcd is a shell function that manipulates a *ksh*(1) environment variable. The optical disk utilities use this variable. To use the *odcd* function, add the following line to the environment file specified by the *ksh*(1) environment variable ENV:

```
. /ip32/od/odenv
```

The *odcd* user must have execute (search) permission in *odpathname*.

Because a new process is created to execute each command, *odcd* would be ineffective if it were written as a normal command; therefore, it is recognized by and is internal to the shell. Optical disk commands use this path name when relative path names are used.

SEE ALSO

odintro(1), odpwd(1), ksh(1).

NAME

odchgrp - change the file group of optical disk files or directories

SYNOPSIS

odchgrp *group odfile* ...
odchgrp *group oddirectory* ...

DESCRIPTION

odchgrp functions identically to its CLIX equivalent, *chgrp*(1), except that *odchgrp* operates on an optical disk.

This utility changes the group ID of files or directories. The group may be either a decimal group ID or a group name found in the group file.

Unless this command is invoked by the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

Only the file owner (or super-user) may change the group for that file.

FILES

/etc/passwd
/etc/group

SEE ALSO

odintro(1), odchmod(1), odls(1), group(4), passwd(4).

NAME

odchmod - change the file protection of optical disk files or directories

SYNOPSIS

odchmod *mode odfile* ...
odchmod *mode oddirectory* ...

DESCRIPTION

odchmod functions identically to its CLIX equivalent, *chmod*(1), except that *odchmod* operates on an optical disk.

The permissions of the named *odfiles* or *oddirectories* are changed according to *mode*, which may be symbolic or absolute. Absolute changes to permissions are stated using octal numbers as follows:

odchmod *nnnn odfile* ...

N is a number from 0 to 7. Symbolic changes are stated using mnemonic characters as follows:

odchmod *xyz,... odfile* ...

X is one or more characters corresponding to *user*, *group*, or *other*; *y* is +, −, or =, signifying permission assignment; and *z* is one or more characters corresponding to permission type.

An absolute mode is given as an octal number constructed from the OR of the following modes:

4000	set user ID on execution
20#0	set group ID on execution if # is **7**, **5**, **3**, or **1** enable mandatory locking if # is **6**, **4**, **2**, or **0**
1000	sticky bit is turned on (see *chmod*(2))
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0040	read by group
0020	write by group
0010	execute (search) by group
0004	read by others
0002	write by others
0001	execute (search) by others

Symbolic changes are stated using letters that correspond both to access classes and to the individual permissions. Permissions to a file may vary depending on the user identification number (UID) or group identification number (GID). Permissions are described in three sequences, each having three characters:

User	Group	Other
rwx	rwx	rwx

This example (meaning that user, group, and others all have read, write, and execute permissions for a given file) demonstrates two categories for granting permissions: the access class and the permissions themselves.

Thus, to change the mode of a file's (or directory's) permissions using *odchmod's* symbolic method, use the following syntax for mode:

[*who*]*operator*[*permission(s)*], ...

A command line using the symbolic method would appear as follows:

```
odchmod g+rw odfile
```

This command would allow group to read and write *odfile*.

Who can be stated as one or more of the following letters:

u	User's permissions.
g	Group's permissions.
o	Other's permissions.
a	Equivalent to **ugo** (all) and is the default if *who* is omitted.

Operator can be + to add *permission* to the *file's* mode, — to take away *permission*, or = to assign *permission* absolutely. (Unlike other symbolic operations, = has an absolute effect in that it resets all other bits.) Omitting *permission* is only useful with = to remove all permissions.

Permission is any compatible combination of the following letters:

r	Read permission.
w	Write permission.
x	Execute permission.
s	Set-user-ID or set-group-ID is turned on.
t	Sticky bit is turned on.
l	Mandatory locking will occur during access.

Multiple symbolic modes separated by commas may be given, although these modes cannot have spaces between them. Operations are performed in the order given. Multiple symbolic letters following a single operator cause the corresponding operations to be performed simultaneously. The letter **s** is only meaningful with **u** or **g**, and **t** works only with **u**.

Mandatory file and record locking (**l**) refers to a file's ability to have its read or write permissions locked while a program is accessing the file. It is not possible to permit group execution and enable a file to be locked on execution at the same time. In addition, it is not possible to turn on the set-group-ID and enable a file to be locked on execution at the same time. Therefore, the following examples are illegal uses and will elicit error messages:

```
odchmod g+x,+l odfile
odchmod g+s,+l odfile
```

Only the owner of a file or directory (or the super-user) may change a file's mode. Only the super-user may set the sticky bit. Before the file's set-group-ID can be turned on, the user's group ID must correspond to the file's and group execution must be set.

EXAMPLES

The following commands deny execution permission to all users. The absolute (octal) example permits only reading permissions.

odchmod a-x *odfile*
odchmod 444 *odfile*

The following commands enable the group and others to read and write a file:

odchmod go=rw *odfile*
odchmod 066 *odfile*

This command causes a file to be locked during access:

odchmod +l *odfile*

These examples enable all to read, write, and execute the file. They also turn on the set-group-ID.

odchmod =rwx,g+s *odfile*
odchmod 2777 *odfile*

SEE ALSO

odintro(1), odls(1).

NAME

odchown - change file ownership of optical disk files or directories

SYNOPSIS

odchown *owner odfile* ...
odchown *owner oddirectory* ...

DESCRIPTION

odchown functions identically to its CLIX equivalent, *chown*(1), except that *odchown* operates on an optical disk.

odchown changes the owner of the *odfiles* or *oddirectories* to *owner*. The owner may be a decimal user ID or a login name in the password file.

Unless the super-user invokes this command, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

Only the owner of the file (or the super-user) may change the owner for that file.

FILES

/etc/passwd
/etc/group

SEE ALSO

odintro(1), odchmod(1), odls(1), group(4), passwd(4).

NAME

odcp - copy optical disk files

SYNOPSIS

odcp *file1* [*file2* ...] *target*

DESCRIPTION

odcp copies files to *target*. It is used to copy files from a magnetic disk to an optical disk, from an optical disk to a magnetic disk, or from one optical disk to another. Either the source file (*file1*), target file (*target*), or both may be an optical disk path specification. If *target* is an existing file on the optical disk, a new copy of the file is created since blocks cannot be rewritten on a Write Once Read Many (WORM) medium.

File1 and *target* can never be the same. If *target* is a directory, one or more files are copied to that directory. If *target* is a file, its contents are destroyed.

If *target* is not a file, a new file is created with the same mode as *file1* except that the sticky bit is not set unless you are super-user; the owner and group of *target* are those of the user. If *target* is a file, copying a file to *target* does not change the file's mode, owner, or group. The last modification time for *target* (and last access time if *target* did not exist) and the last access time of *file1* are set to the time the copy was made.

SEE ALSO

odintro(1), odchmod(1), odln(1), odrm(1).

WARNINGS

Be careful when using shell metacharacters.

NAME

oddf - report number of free blocks and i-nodes on an optical volume

SYNOPSIS

oddf [**-t**] [**-f**] *volume*

DESCRIPTION

oddf displays the number of free blocks and i-nodes in the mounted volume specified by *volume* by examining the counts kept in the super-blocks.

The *oddf* command uses the following options:

-t Report both the total allocated blocks and i-nodes, and the total allocated free blocks and i-nodes.

-f Report free blocks.

SEE ALSO

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

NAME

odintro - introduction to the optical disk file system

DESCRIPTION

The optical disk file system is implemented for the Write Once Read Many (WORM) optical disk media. It emulates the design of the UNIX System V file system, though it is not mounted as a normal file system. It can be accessed only through utilities converted to use this file system. Many of the basic UNIX file system utilities have been converted, as well as *fmu*(1).

An optical disk platter is a two-sided storage medium. A volume is one side of a platter. Volumes are not partitioned. Each volume contains a root directory and a tree of user-defined subdirectories.

Files are referenced on an optical disk file system through an optical disk path specification. The specification consists of a colon (:), volume name, colon, and path name. For example, an absolute optical disk path specification could be as follows:

> *:volume:/directory1/directory2/basename*

A volume name can have 1 to 16 characters. These characters may come from the set of all character values excluding colon, \0 (null), and slash.

Absolute path specifications must specify the volume name and the path name must begin with a slash. Relative path specifications may be used if *odcd*(1) was used to define a default optical disk directory. A relative path specification omits the volume name in the specification and the path name is not required to begin with a slash. For example, a relative optical disk path specification could be as follows:

```
::/
::file
::.
```

Regardless of whether the volume name is specified, the colons that would surround it must be present. If a path name begins with a slash, the path search begins at the volume's root directory. Otherwise, the search begins at the current default optical disk directory. A path name consisting of a slash by itself names the volume's root directory.

In other optical disk documentation, *odpnathname* refers to an optical disk path specification. This specification can be a file or directory. *Odfile* refers to an optical disk file and *oddirectory* refers to an optical disk directory.

Directory entries are called links. By convention, a directory contains at least two links, **.** and **..**, referred to as dot and dot-dot, respectively. Dot is the directory itself and dot-dot is its parent directory. Thus, **::.** is the current default optical disk directory and **::..** is its parent.

The optical disk file system is accessed in two different environments: standalone and jukebox. In the standalone environment, an optical disk platter must be manually loaded in an optical disk drive. The system administrator must then mount a volume on that platter so that the optical disk utilities may then access it. In the jukebox environment, an optical disk jukebox will mount optical disk platters as needed by *fmu*(1) or at the request of the system administrator.

If an optical disk path specification is used in the *fmu*(1) commands send, receive, rcd, or cd, the volume implied by the specification will automatically be loaded in a drive and mounted if the volume resides in a jukebox. An *fmu*(1) session can mount only one volume at a time. Also, as soon as *fmu*(1) exits, the volume is unmounted and unloaded. If the volume is in a standalone environment, the system administrator must have already mounted the volume using the *odmount*(1M) command.

SEE ALSO

odcd(1), odchgrp(1), odchmod(1), odchown(1), odcp(1), oddf(1), odln(1), odls(1), odmkdir(1), odmv(1), odpwd(1), odrm(1), odrmdir(1), jbconfig(1), JBCFG(4), STANDCFG(4).
odfsck(1M), odlabel(1M), odmount(1M), odreadlabel(1M), odumount(1M), jbexport(1M), jbimport(1M), jbinventory(1M), jblabel(1M), jbstart(1M), jbterminate(1M), jbvaryoff(1M), jbvaryon(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

odls - list contents of optical disk directories

SYNOPSIS

odls [-RadCxmlnogrtucpFbqisf] [*odpathname*]

DESCRIPTION

odls functions identically to its CLIX equivalent, *ls*(1), except that *odls* lists files in an optical disk directory.

For each directory argument, *odls* lists the directory contents; for each file argument, *odls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format is to list one entry per line. The **-C** and **-x** options enable multicolumn formats, and the **-m** option enables stream output format. To determine output formats for the **-C**, **-x**, and **-m** options, *odls* uses the environment variable COLUMNS to determine the number of character positions available on one output line. If this variable is not set, the *terminfo*(4) database is used to determine the number of columns, based on the environment variable TERM. If this information cannot be obtained, 80 columns are assumed.

The following options are available:

-R Recursively list subdirectories encountered.

-a List all entries, including those that begin with a dot (.), which are normally not listed.

-d If an argument is a directory, list only its *odpathname* (not its contents); often used with -l to obtain the directory's status.

-C Print multicolumn output with entries sorted down the columns.

-x Print multicolumn output with entries sorted across rather than down the page.

-m Print in stream output format; files are listed across the page, separated by commas.

-l List in long format giving mode, number of links, owner, group, size in bytes, and last modification time for each file (see below). If the file is a special file, the size field will contain the major and minor device numbers rather than a size.

-n Same as -l except that the owner's UID and group's GID numbers are printed, rather than the associated character strings.

-o Same as -l except that the group is not printed.

-g	Same as -l except that the owner is not printed.
-r	Reverse the sort order to get reverse alphabetic or oldest first as appropriate.
-t	Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See **-n** and **-c**.)
-u	Use the last access time instead of modification time for sorting (with the **-t** option) or printing (with the **-l** option).
-c	Use the last i-node modification time (such as when a file was created or a mode changed) for sorting (**-t**) or printing (**-l**).
-p	Put a slash (/) after each file name if the file is a directory.
-F	Put a slash (/) after each file name if the file is a directory and put an asterisk (*) after each file name if the file is executable.
-b	Force nongraphics characters in file names to be printed in the octal **\ddd** notation.
-q	Force nongraphics characters in file names to be printed as the character **?**.
-i	For each file, print the i-number in the first column of the report.
-s	Give size in blocks (including indirect blocks) for each entry.
-f	Force each argument to be interpreted as a directory and list the name found in each slot. This option turns on **-s**; the order is the order in which entries appear in the directory.

The mode printed under the **-l** option consists of ten characters. The first character may be one of the following:

d	The entry is a directory.
b	The entry is a block special file.
c	The entry is a character special file.
p	The entry is a fifo ("named pipe") special file.
—	The entry is an ordinary file.

The next nine characters are interpreted as sets of three bits each. The first set refers to the owner's permissions; the next set refers to permissions of others in the user group of the file; the last set refers to all others. Within each set, the three characters indicate permission to read, write, and execute the file as a program, respectively. Execute permission for a directory is permission to search the directory for a specified file.

odls -l prints its output as follows:

```
-rwxrwxrwx  1 smith  dev     10876  May 16 9:42 part2
```

This horizontal configuration provides a lot of information. Reading from right to left, it is seen that the current directory holds one file, named **part2**. Next, the last time the file's contents were modified was 9:42 AM on May 16. The file is moderately sized, containing 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group **dev**, and the login name is **smith**. The number (in this case **1**) indicates the number of links to file **part2**. Finally, the row of dashes and letters shows that user, group, and others have permission to read, write, and execute **part2**.

The execute (**x**) symbol occupies the third position of the three-character sequence. A — in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

r	The file is readable.
w	The file is writable.
x	The file is executable.
—	The indicated permission is not granted.
l	Mandatory locking will occur during access. (The set-group-ID bit is on and the group execution bit is off.)
s	The set-user-ID or set-group-ID bit and the corresponding user or group execution bit are on.
S	Undefined bit-state. (The set-user-ID bit is on and the user execution bit is off.)
t	The 1000 (octal) bit, or sticky bit (see *chmod*(1), and the execution bit are on.
T	The 1000 bit is on and execution is off (undefined bit-state).

For user and group permissions, the third position is sometimes occupied by a character other than **x** or **—**. **s**, referring to the state of the set-ID bit (the user's or the group's), may also occupy this position. For example, the ability to assume the same ID as the user during execution is used during login when the user begins as root but needs to assume the identity stated at login.

In the sequence of group permissions, **l** may occupy the third position. **l** refers to mandatory file and record locking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For other permissions, the third position may be occupied by **t** or **T**. These refer to the state of the sticky bit and execution permissions.

EXAMPLES

This example describes a file that the user can read, write, and execute and that group and others can read:

```
-rwxr--r--
```

This example describes a file that the user can read, write, and execute; the group and others can read and execute it. This permission allows the user presently executing it to assume its user ID during execution:

```
-rwsr-xr-x
```

This example describes a file that only the user and group can read and write and that can be locked during access:

```
-rw-rwl---
```

This command will print the names of all files in the current directory, including those that begin with a dot (.), which normally do not print:

```
odls -a
```

This command will provide information such as all files (including non-printing ones (**a**)); the i-number, the memory address of the i-nodes associated with the file, printed in the left-hand column (**i**); and the size of the files (in blocks) printed in the column to the right of the i-numbers (**s**). The report is printed in the numeric version of the long list, printing the UID (instead of user name) and GID (instead of group name) numbers associated with the files.

```
odls -aisn
```

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	UIDs for **ls -l** and **ls -o**
/etc/group	GIDs for **ls -l** and **ls -g**
/usr/lib/terminfo/?/*	terminal information database

SEE ALSO

odintro(1), odchmod(1).

BUGS

Unprintable characters in file names may confuse the columnar output options.

NAME

odmkdir - create optical disk directories

SYNOPSIS

odmkdir [**-m** *mode*] [**-p**] *oddirectory*

DESCRIPTION

odmkdir creates the specified directory on the volume implied by the optical disk directory name *oddirectory*. Default permissions as defined by *umask*(1) are set on the directory unless the **-m** option is specified.

Standard entries in a directory (such as the files **.** for the directory itself and **..** for its parent) are created automatically. *odmkdir* cannot create these entries by name. Directory creation requires write permission in the parent directory.

The owner ID and group ID for the new directories are set to the process's user ID and group ID, respectively.

The following options are available:

-m *mode* Create the directory with the specified protection mode. An absolute mode is given as an octal number constructed from the OR of the following modes:

4000	set user ID on execution
20#0	set group ID on execution if # is **7**, **5**, **3**, or **1** enable mandatory locking if # is **6**, **4**, **2**, or **0**
1000	sticky bit is turned on (see *chmod*(2))
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0040	read by group
0020	write by group
0010	execute (search) by group
0004	read by others
0002	write by others
0001	execute (search) by others

-p Create the last directory in the path name and any parent directories in the path that do not exist.

EXAMPLES

Use the following command line to create the subdirectory structure **/ltr/jd/jan** on volume **development**:

```
odmkdir -p :development:/ltr/jd/jan
```

SEE ALSO

odintro(1), odrm(1), intro(2).
sh(1), umask(1) in the *UNIX System V User's Reference Manual.*

DIAGNOSTICS

odmkdir returns exit code 0 if all directories given in the command line were created successfully. Otherwise, it displays a diagnostic message and returns a nonzero exit code.

NAME

odmv - rename optical disk files or directories

SYNOPSIS

odmv [**-f**] *odpathname* ... *odtarget*

DESCRIPTION

odpathname is moved to *odtarget*. *Odpathname* and *odtarget* can never be the same. If *odtarget* is a directory, one or more files are moved to that directory. If it is a file, its contents are destroyed.

If *odmv* determines that the target mode forbids writing, it prints the mode (see *odchmod*(1)), prompts the user, and reads standard input for one line. If the line begins with **y**, the move occurs (if permissible); if not, the command exits. When the **-f** option is used or if the standard input is not a terminal, no questions are asked and the move is performed.

odmv will allow *odpathname* to be a directory. The directory is renamed only if the two directories have the same parent; *odpathname* is renamed *odtarget*. If *odpathname* is a file and *odtarget* is a link to another file with links, the other links remain and *odtarget* becomes a new file.

SEE ALSO

odintro(1), odchmod(1), odcp(1), odln(1), odrm(1).

WARNINGS

If *odpathname* and *odtarget* are on different volumes, *odmv* must copy the file and delete the original. In this case, any linking relationship with other files is lost.

Be careful when using shell metacharacters.

NAME

odpwd - display the current default directory used by optical disk commands

SYNOPSIS

odpwd

DESCRIPTION

odpwd displays the current optical disk default path name as defined by *odcd*(1). The optical disk commands use this path name when relative path names are used.

SEE ALSO

odintro(1), odcd(1).

DIAGNOSTICS

"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred to a CLIX system administrator. *odfsck*(1M) may be used to correct the problem.

NAME

odrm - delete optical disk files

SYNOPSIS

odrm [**-f**] [**-i**] *odfile* ...
odrm -r [**-f**] [**-i**] *oddirectory* [*odfile* ...]

DESCRIPTION

odrm removes the entries for one or more files from a directory. If an entry is the last link to the file, the file can no longer be accessed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file is printed followed by a question mark. There is a prompt for confirmation. If the answer begins with y (for yes), the file is deleted. Otherwise, the file remains. If the standard input is not a terminal, the command operates as if the **-f** option is in effect.

Three options apply to *odrm*:

-f Remove all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (regardless of individual file permissions). If the user attempts to remove a write-protected directory, the **-f** option does not suppress an error message.

-r Recursively remove all directories and subdirectories in the argument list. Files in the directory will be removed and then the directory will be removed. The user is normally prompted to remove any write-protected files the directory contains. However, if the **-f** option is used or if the standard input is not a terminal and the **-i** option is not used, write-protected files are removed without prompting.

When removal of a nonempty, write-protected directory is attempted, the command will always fail, resulting in an error message.

-i Interactively confirm removal of each file and directory. This option overrides the **-f** option and remains in effect even if the standard input is not a terminal.

SEE ALSO

odintro(1), odln(1), odcp(1), odmv(1).

DIAGNOSTICS

All messages are self-explanatory. Removing the files **.** and **..** is forbidden to avoid the consequences of inadvertently making the following type of mistake:

```
odrm -r ::.*
```

odrm returns exit codes of 0 if all specified directories are removed successfully. Otherwise, a nonzero exit code is returned.

NAME

odrmdir - delete optical disk directories

SYNOPSIS

odrmdir [**-p**] [**-s**] *oddirectory* ...

DESCRIPTION

odrmdir removes only empty directories.

Two options apply to *odrmdir*:

-p Remove the specified directories and their parent directories that become empty. A message is printed on standard output explaining whether the whole path was removed or part of the path remains.

-s Suppress the message printed when a standard error occurs and when **-p** is in effect.

SEE ALSO

odintro(1), odmkdir(1).

DIAGNOSTICS

All messages are generally self-explanatory. *odrmdir* returns an exit code of 0 if all specified directories are removed successfully. Otherwise, it returns a nonzero exit code.

NAME

odt - examine and modifies files

SYNOPSIS

odt *file*

DESCRIPTION

odt is an interactive program that allows the user to examine and modify the contents of a file. Both location pointers and contents are displayed in hexadecimal format. Input is also interpreted as hexadecimal.

The commands available within *odt* are:

/	Display the current location within the file and the contents of that location.
addr/	Set the current location pointer equal to *addr* and display this location.
y	Set display to byte mode and display current location.
w	Set display to word mode and display current location.
l	Set display to long word mode and display current location.
value<CR>	Set the contents of the current location equal to *value*.
<LF>	Increment the current location pointer and display location.
^	Decrement the current location pointer and display location.
*value***s**	Search for *value* in the file starting at the current location and display location.
s	Continue the search for the last number searched for starting at the current location.
<ESC>	Exit from *odt*.

All changes are made to *file* immediately, and there is no backup mechanism for restoring unwanted changes.

WARNINGS

odt is dangerous when used with special files.

NAME

pc - Pascal compiler

SYNOPSIS

pc [*option* ...] *file* ...

DESCRIPTION

pc is the interface to the Green Hills Pascal compiler. Files ending in **.p** are assumed to be Pascal source files and are compiled to relocatable object files whose names are derived by replacing the **.p** suffix of the source file name with **.o.** If no compilation errors are detected, *pc* will attempt to link the relocatable objects to produce an executable file. If the linking is successful, the intermediate object files are deleted.

pc works similarly on files with a **.s** suffix. In this case, however, the files are assumed to be assembly source and the compilation phase is bypassed.

pc will also accept other file types or combinations of file types as input. It will compile or assemble files that end in **.p** or **.s** and pass the results and/or other file names to the link editor. When invoking the link editor, *pc* will specify **-lpc**, **-lm**, and **-lc** on the link editor command line.

An executable may be built from separately-compiled source files. However, only one of the source files may contain a **program** statement. Variables may be shared among source files by declaring them at the outermost level in the files in which they are referenced. A function or procedure defined in one file may be called from another file provided the caller declares the function or procedure with an **external** declaration. The syntax for an **external** declaration is identical to the syntax of a **forward** declaration.

Pascal object files may be linked with C and/or FORTRAN object files. Pascal passes parameters by value unless the **var** keyword is used in the formal parameter declaration. In this case, parameters are passed by reference. C always passes parameters by value, FORTRAN always passes by reference. Note that all FORTRAN function, subroutine, and common names are stored in the symbol table with an underscore (_) appended.

A list of command line options follows.

-c Suppress the link edit phase of the compilation and force an object file to be produced even if only one program is compiled.

-g Cause the compiler to generate additional information needed for the use of source language debuggers like *sdb*(1).

-C Enable run-time checking of subranges and array bounds.

-ga Generate a frame pointer for stack traces.

-o *file-name*
Place the executable binary output from the link edit phase in the file named *file-name*. If this option is not specified, the executable file will be named **a.out**. This option is ignored if **-c** or **-S** is present.

-O Optimize the program for speed at the expense of code space. The default compiler settings cause Green Hills Pascal to perform most or all optimizations that other compilers perform only under the **-O** option. Experiment with this option to determine whether the additional code size and compilation time are worth the execution speed gain.

-p Arrange for the compiler to produce code that counts the number of times each routine is called; also, if link editing occurs, replace the standard startoff routine with one that automatically calls *monitor*(3C) at the start and arranges to write out a **mon.out** file at normal object program termination. An execution profile can then be generated by using *prof*(1).

-s Compile the program in ANSI-compatible mode. Generate errors when extensions to the ANSI Pascal standard are used. Using this option will change the default subrange of the set type from 0-31 to 0-255, resulting in poorer code for set handling.

-S Compile the named Pascal programs and leave the assembler-language output on corresponding files suffixed with **.s**. The assembler and link edit phases are suppressed.

-v Print the program name and command line arguments of each phase of the compilation/link process.

-w Suppress warning diagnostics.

-X *n* Turn on compile-time option number *n*.

-Z *n* Turn off compile-time option number *n*. The available compile-time options are listed below.

- **9** Disable the local (peephole) optimizer.
- **18** Do not allocate programmer-defined local variables to a register unless they are declared **register**.
- **32** Display the names of files as they are opened. This is useful for determining why the compiler cannot find an include file.
- **37** Emit a warning when dead code is eliminated.
- **39** Do not move frequently-used procedure and data addresses to registers.
- **58** Do not put an underscore in front of the names of global variables and procedures. This option is not recommended because it produces symbols that are incompatible with the rest of the CLIX System.
- **59** Turn off case sensitivity.
- **87** Disable the optimization that deletes all code that stores in or modifies variables that are never read from.

89	Pack structures with no space between members. WARNING: This may make the structure members impossible to access.
156	Export the names of variables declared in the outermost scope of a Pascal main program for use in other modules of a multiple-module executable. The default is for variables declared in the outer scope of a Pascal main program to be static and inaccessible from other modules.
168	Do not move invariant floating-point expressions out of loops.
174	Append an underscore to the names of all external procedures and functions to avoid name conflicts with library routines. Set by default if **-s** is specified.
190	Assume halfword objects are not aligned.
191	Assume word objects are not aligned.
192	Assume single-precision objects are not aligned.
193	Assume double-precision objects are not aligned.
194	Assume word objects are aligned only to halfword boundaries.
195	Assume single-precision objects are aligned only to halfword boundaries.
196	Assume double-precision objects are aligned only to half word boundaries.
197	Assume double-precision objects are aligned only to word boundaries.

FILES

file.p	Pascal source input file
file.o	object file; generated or input
/usr/lib/pcom, lib/pcom	Pascal compiler
/bin/as	assembler, *as*(1)
/bin/ld	link editor, *ld*(1)
/lib/crt[1n].o	run-time startoff
/lib/mcrt[1n].o	profiling startoff
/usr/lib/libpc.a, /lib/libpc.a	Pascal intrinsic functions and I/O library
/lib/libc.a	standard C library, see sections (3C) and (3S) in the *UNIX System V Programmer's Reference Manual*
/lib/libp/lib*.a	profiled versions of libraries

SEE ALSO

adb(1), as(1), ld(1), sdb(1).
prof(1) in the *UNIX System V Programmer's Reference Manual.*
The Greenhills Software Users Manual Pascal-CLIPPER.

NAME

qdel - delete or signal NQS requests

SYNOPSIS

qdel [**-k**] [*-signo*] [**-u** *user-name*] *request-id*[*@host*] ...

DESCRIPTION

qdel deletes or signals the Network Queuing System (NQS) requests specified by *request-ids*. If @*host* is specified, the *host* is deleted. Queued and waiting requests are deleted. Running requests are signaled if the **-k** or the *-signo* options are used. The **-k** option will send the SIGKILL signal and the *-signo* option will send the signal associated with *signo* to the specified requests. Routing, arriving, and departing requests are not affected.

To delete or signal an NQS request, the invoking user must be the owner of the request. The **-u** option, however, provides a way to avoid this rule. The **-u** option specifies requests owned by the user *user-name*. This option may be used only if the invoking user is the super-user or has NQS operator privileges. If a *request-id* that is not owned by *user-name* is specified, an error message will be generated.

Request-id uniquely identifies an NQS request regardless of where the request is in the network of NQS machines. *Request-id* has the form *seqno*[*.host-name*]. *Seqno* identifies the sequence number assigned to the request on the originating host. *Host-name* identifies the originating host. If the *host-name* portion of a *request-id* is omitted, the local host is assumed.

The *request-id* of an NQS request is displayed when the request is first submitted (unless the *silent* mode of operation is specified). The user can also obtain the *request-id* of any request by using the *qstat*(1) command.

SEE ALSO

qdev(1), qlimit(1), qpr(1), qstat(1), qsub(1), setpgrp(2), signal(2).
qmgr(1M) in the *CLIX System Administrator's Reference Manual.*
kill(1) in the *UNIX System V Programmer's Reference Manual.*

WARNINGS

When an NQS request is spawned, a new process group is established for all processes in the request. If the **-k** or *-signo* option is used, a signal will be sent to all processes in the process group. However, if a process successfully executes a *setpgrp*(2) call, it will not receive any signals sent by *qdel*. The *kill*(1) command may be used to delete such processes.

NAME

qdev - display the status of NQS devices

SYNOPSIS

qdev [*device-name*[@*host*] ...]

DESCRIPTION

qdev displays the status of Network Queuing System (NQS) devices. *qdev*, without any arguments, displays the current status of all NQS devices on the local host. Otherwise, *qdev* displays the status of the devices specified by *device-name*. If @*host* is specified, the device located on *host* is displayed. The device is assumed to be on the local machine unless a particular host is specified by @*host*.

A device header with the following format is displayed for each of the specified devices:

> *device-name*@*host-name*
> **Fullname:**
> **Server:**
> **Forms:**
> **Status – [ ];**

"Fullname:" lists the full path name of the special file associated with the device. "Server:" lists the command line that will be used to *execve*(2) the device server. "Forms:" lists the forms configured for the device. "Status:" displays the general state of the device.

The general state of a device is defined by two principal properties. The first property is whether the device is willing to continue accepting queued requests. If it is willing, the state of the device is ENABLED. If the device is unwilling to continue accepting queued requests and is idle, its state is DISABLED. The state of the device is ENABLED/CLOSED if the device is unwilling to continue accepting queued requests but is not yet idle.

The second principal property of a device is whether the device is busy. If the device is busy, it is in an ACTIVE state. If the device is idle and not out of service, it is in an INACTIVE state. If the device is idle and out of service, it is in a FAILED state. The FAILED state covers both hardware and software failures.

If a device is busy, information about the active request follows the device header. The name of the request, the ID of the request, and the name of the request owner are displayed.

SEE ALSO

qdel(1), qlimit(1), qpr(1), qstat(1), and qsub(1).
qmgr(1M) in the *CLIX System Administrator's Reference Manual.*

CAVEATS

qdev does not currently support requests to display the status of devices on remote hosts. If the specified host is not the local host, an error will be generated.

NAME

qlimit - show supported batch limits and shell strategy for the local host

SYNOPSIS

qlimit

DESCRIPTION

qlimit displays the batch request resource limits that can be directly enforced on the local host and the batch request shell strategy for the local host.

Network Queuing System (NQS) supports many batch request resource limits that can be applied to an NQS batch request. However, this implementation does not support the entire set of limits that NQS provides. The limits supported are a per-process nice value and a per-process file size.

The limits applied to a batch request are always restricted to the limits that can be directly supported by the underlying implementation. If a batch request specifies a limit that cannot be enforced by the underlying implementation, the limit is ignored and the batch request will operate as though a limit (other than the physical maximums) had not been placed on that resource.

When an attempt is made to queue a batch request, each limit specified by the request (that can also be supported by the local implementation) is compared to the corresponding limit for the destination batch queue. If a limit for a batch queue is defined to be "unlimited" or greater than or equal to the corresponding limit of the batch request, the request will be successfully queued (barring other abnormal conditions). If a request specifies a limit of "infinity", the corresponding limit for the queue must also be "infinity".

The limit checks are performed regardless of whether the batch request was submitted by directly using the *qsub*(1) command or by indirectly placing the request in a pipe queue. It is impossible for a batch request to be queued in an NQS batch queue if any of these limit checks fail.

If a request does not specify a limit that is supported on the local host, the corresponding limit as configured for the destination queue becomes the limit for the request.

Upon the successful queuing of a request in a batch queue, the limits under which the request executes are frozen and are not modified by subsequent *qmgr*(1M) commands that alter the limits of the containing batch queue.

As mentioned above, this command also displays the shell strategy as configured for the local host. Without a shell specification for a batch request, NQS must choose the shell that should be used to execute the request. NQS supports three different strategies to solve this problem: *fixed*, *free*, and *login*.

A fixed shell strategy means that all batch requests will be executed using the shell chosen by the system administrator.

A free shell strategy means that the batch requests will be run the same as an interactive invocation of the request would be run.

A login shell strategy means that the batch requests will be executed by the user's normal login shell.

The default shell strategy is free. Hosts machines which reach the maximum number of process's allowed on the system should use a fixed or login strategy because a single shell will be exec'd to run all commands in the request script. *qlimit* will display the chosen shell if the fixed strategy has been selected.

SEE ALSO

qdel(1), qdev(1), qpr(1), qstat(1), qsub(1).
qmgr(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

qpr - submit a hardcopy print request to NQS

SYNOPSIS

qpr [*option* ...] [*file* ...]

DESCRIPTION

qpr places the named files in a Network Queuing System (NQS) queue to be printed by a device such as a line printer or a laser printer. If a file is not specified, *qpr* reads from **stdin**.

NQS has queue access restrictions. For each queue with a queue type other than network, access may be either unrestricted or restricted. If access is unrestricted, any request may enter the queue. If access is restricted, a request can enter the queue only if the requester or the requester's login group has access to that queue (see *qmgr*(1M)). Requests submitted by the super-user are an exception; they are always queued, even if the super-user has not explicitly been given access. *qstat*(1) may be used to determine who has access to a particular queue.

qpr prints a request ID to **stdout** when a request is queued successfully. This request ID can be compared with what is reported by *qdev*(1) and *qstat*(1) to learn the outcome of a request. It can also be given as an argument to *qdel*(1) to delete a request. A request ID has the form *seqno.host-name*, where *seqno* refers to the sequence number assigned to the NQS request, and *host-name* refers to the name of the originating machine. This identifier is used throughout NQS to uniquely identify the request anywhere in the network.

The following options are available and may be intermixed with file names.

-a *date-time*

Submit at the specified date and/or time. When this option is not specified, *qpr* submits the request immediately.

If a *date-time* specification is composed of two or more tokens separated by white space characters, the *date-time* specification must be enclosed in quotation marks as in **"-a July, 4, 2026 12:31-EDT"**. If not specified in quotation marks, the specification should be escaped so that the shell will interpret the *date-time* specification as a single lexical token.

The syntax accepted for the *date-time* parameter is flexible. Unspecified date and time values default to an appropriate value. (For example, if a date is not specified, the current month, day, and year are assumed.)

A date can be specified as a month and a day (current year assumed). The year can also be explicitly specified. It is also possible to specify the date as a weekday name (such as **Tues**), or as one of the strings **today** or **tomorrow**. Weekday and month names can be abbreviated by any three-character (or longer) prefix to the actual name.

An optional period can follow an abbreviated month or day name.

Time of day specifications can be given using a 24-hour clock or am and pm specifications may be used alternatively. When a meridian is not specified, a 24-hour clock is assumed.

The time of day specification is interpreted using the precise meridian definitions. **12am** refers to the 24-hour clock time of 0:00:00; **12m** refers to noon; and **12-pm** refers to 24:00:00. Alternatively, the phrases **midnight** and **noon** are accepted as time of day specifications, where **midnight** refers to 24:00:00.

A time zone may also appear at any point in the *date-time* specification. Thus, "**April 1, 1987 13:01-PDT**" is a legal specification. When a time zone is not specified, the local time zone is assumed, with daylight savings time being inferred when appropriate based on the date specified.

Not all alphabetic comparisons are case-sensitive. Both **WeD** and **weD** refer to Wednesday.

Examples of valid *date-time* specifications are as follows:

```
"01-Jan-1986 12am, PDT"
"Tuesday, 23:00:00"
"11pm tues."
"tomorrow 23-MST"
```

-d *name=value*
: Define an environment variable *name* with the given *value* to be put in the environment of the device server. This option is specific to devserver.

-e *tag=filename*
: Associate *tag* with the *filename*, export the file to the server machine, and put *tag* in the environment of the device server so that the server can access the ancillary file by looking at the environment variable *tag*. This option is specific to devserver.

-f *form-name*
: Limit the set of acceptable devices to devices that are loaded with the form *form-name*. When this option is not specified, *qpr* submits the request only to a device loaded with the default form. If a default form is not defined, the request is submitted to the first available output device without regarding the forms configured for the device. In any case, only devices associated with the chosen queue are considered.

-l *message*
: Log *message* in the device accounting file if device accounting is turned on at the destination device queue. This option is specific to devserver.

-mb Send mail to the user on the originating machine when the request begins execution. If the **-mu** option is also present, mail is sent to the user specified by the **-mu** option instead of to the invoking user.

-me Send mail to the invoking user on the originating machine when the request has ended execution. If the **-mu** option is also present, mail is sent to the user specified by the **-mu** option instead of to the invoking user.

-mu *user-name*
Specify that any mail concerning the request should be delivered to the user *user-name*. *User-name* has the form *user*[@*machine*]. When this option is not specified, any mail concerning the request is sent to the invoker on the originating machine.

-n *number-of-copies*
Print *number-of-copies* copies. The default is 1.

-o *options*
Place the following *options* on the end of the argument list of the device filter before it is started to process the output file. Since *options* is one argument, multiple options must be quoted as one argument. This option is specific to devserver.

-p *priority*
Assign an intraqueue priority to this request. The specified *priority* must be an integer, and must be in the 0-63 range, inclusive. A value of 63 defines the highest intraqueue request priority, while a value of 0 defines the lowest. This priority does not determine the execution priority of the request. This priority is used only to determine the relative ordering of requests within a queue.

When a request is added to a queue, it is placed at a specific position within the queue so that it appears ahead of all existing requests with priorities less than the priority of the new request. Similarly, all requests with a higher priority remain ahead of the new request when the queuing process is complete. When the priority of the new request equals the priority of an existing request, the existing request takes precedence over the new request.

If the user does not choose an intraqueue priority, the value configured by the system administrator will be used. If a value has not been configured by the system administrator, a default value of 31 is assigned to the request.

-q *queue-name*
Specify the queue to which the device request is to be submitted. If the **-q** *queue-name* specification is not given, the user's environment is searched for the variable QPR_QUEUE. If this environment variable is found, the character string value for QPR_QUEUE is presumed to be the name of the queue to which the request should be submitted. If the QPR_QUEUE environment variable is not found, the request is

submitted to the default device request queue if one has been defined by the local system administrator. Otherwise, the request cannot be queued and an appropriate error message is displayed.

-r *request-name*
: Assign a name to this request. When the -r option is not specified, the *request-name* defaults to the name of the first print file (with the leading path name removed) specified on the command line. If a print file was not specified, the default *request-name* assigned to the request is **stdin**.

  In all cases, if the *request-name* begins with a digit, the character **R** is prefixed to prevent a *request-name* from beginning with a digit. All *request-names* are truncated to a maximum length of 15 characters.

-R
: Delete the original files after NQS is finished with them. This normally means that the original files are deleted immediately after the files are successfully copied into the spool directory.

  If the **-s** is specified with the **-R** option, the original files will not be deleted until one of two events occurs. If the file is printed/plotted locally, the original files are deleted after the job has completed. If the request is routed to a remote machine, the files are deleted after the request has been transferred to the remote machine.

  The **-R** option is intended for use with temporary files. Requests that are deleted or aborted will also cause the original files to be deleted.

-s
: Symbolically link the files into the NQS spool directories rather than copying them. If the **-s** option is used, files submitted should not be renamed, moved, or deleted until the device request has left the machine or has completed printing. Using the **-s** option speeds up the submission of very large files because they are not copied into the spool directory.

-t *type*
: Specify that the format of the data is *type*. This option is specific to devserver.

-x
: When a device request is submitted, the following environment variables are automatically defined in the environment of the device server: QPR_HOST, QPR_REQID, QPR_REQNAME, and QPR_QUEUE. These environment variables refer to the host name the request originated from, the request ID, the request name, and the name of the queue the request eventually executes in. If the **-x** option is specified, all remaining environment variables are exported to the environment of the device server. This option is specific to devserver.

-z
: Submit the request silently. If the request is submitted successfully,

nothing will be written to **stdout** or **stderr**.

SEE ALSO

qdel(1), qdev(1), qlimit(1), qstat(1), qsub(1).
qmgr(1M) in the *CLIX System Administrator's Reference Manual.*
mail(1) in the *UNIX System V User's Reference Manual.*

NAME

qstat - display the status of NQS queues

SYNOPSIS

qstat [**-a**] [**-b**] [**-d**] [**-l**] [**-m**] [**-p**] [**-r**] [**-u** *user-name*] [**-x**] [*queue-name*[@*host-name*] ...]

qstat [**-c**] [*complex-name* ...]

DESCRIPTION

qstat displays the status of Network Queuing System (NQS) queues. *qstat*, without a *queue-name* argument, displays the current status of all NQS queues on the local host. Otherwise, *qstat* displays the status of the queues specified by *queue-name*. The queue is assumed to be on the local machine unless a particular host is specified by @*host-name*.

qstat normally displays information only about requests in the specified queues owned by the invoker. This may be changed by one of the following options.

-a	Display the status of all requests in the queue.
-b	Restrict the display to batch queues.
-d	Restrict the display to device queues.
-p	Restrict the display to pipe queues.
-r	Recursively display pipe queue destinations. After a pipe queue is displayed, display each of the respective queues appearing in the pipe queue's destination list. This option should be used with the **-b**, **-d**, or **-p** option to limit the queues displayed.
-u *user-name*	Display the status only about requests owned by *user-name*.

For each specified queue, a queue header is displayed. The queue header displays the queue name, queue type, queue state, an indication of whether the queue accepts requests only from pipe queues, and the number of requests in the queue. Additional information about the queue may be obtained with the **-x** (extended format) option. This option will display the queue's priority, run limit, access restrictions, cumulative use statistics, server and destinations (if a pipe queue), queue-to-device mappings (if a device queue), and resource limits (if a batch queue).

The general state of a queue is defined by two principal properties. The first property determines whether requests can be submitted to the queue. If they can and the local NQS daemon is present, the state of the queue is ENABLED. If the local daemon is not present, the queue is in a CLOSED state. If a request cannot be submitted, it is in a DISABLED state. Requests can be submitted only when the queue is in the ENABLED state.

The second principal property of a queue determines if requests that are ready to run but are not running will be allowed to run when running requests complete. It also determines whether any requests are running in the queue. If queued requests are blocked and no requests are running, the queue is in a STOPPED state. If queued requests are blocked and at least one request is running, the queue is in a STOPPING state. In this state, requests that are running will be allowed to complete. However, no new requests will be spawned.

If the NQS daemon prevents queued requests from running and at least one request is running, the queue is in a RUNNING state. If the daemon prevents queued requests from running and no requests are running, the queue is in an INACTIVE state. If the daemon is not running but the queue would otherwise be in the RUNNING or INACTIVE state, the queue is in a SHUTDOWN state.

Following each queue header, information about requests in the queue is displayed. For each request, the following information is displayed: the request name, the request ID, the request owner, the relative request priority, the current request state, the process group (if the request is running), and the request size (if a device queue). Additional information may be obtained with one of the following options:

-l	Display information about requests in long format.
-m	Display information about requests in medium-length format.
-c	Display information about queue complexes.

The **-l** option displays the time when the request was created, an indication of whether mail will be sent, where mail will be sent, the user name on the originating machine, and the requested forms (if a device queue). If the queue is a batch queue, resource limits, planned disposition of **stderr** and **stdout**, advice concerning the command interpreter, and the *umask*(2) value are also displayed.

The **-m** option displays the time and date the request will run.

The **-c** option displays information about queue complexes and ignores all other options. For each queue complex, the run limit and a listing of the member queues in the complex is shown.

The disposition of a request defines the state of the request. If it is being queued from a remote host, the state of the request is ARRIVING. If it is submitted with a time constraint that has not yet arrived, its state is WAITING. If it is eligible to proceed to a ROUTING or a RUNNING state, it is in a QUEUED state. If it is at the head of a pipe queue and is receiving service there, it is in a ROUTING state. If it departed from a pipe queue and has not yet arrived at its destination, it is in a DEPARTING state. If it reached its destination and is executing, it is in a RUNNING state.

EXAMPLES

A batch request originating on a workstation and destined for the batch queue of a remote machine to be run immediately would first undergo the states, QUEUED, ROUTING, and DEPARTING, in a local pipe queue. The request would then leave the pipe queue and be received by a batch queue on the remote machine. Here, it undergoes the states, ARRIVING, QUEUED, and RUNNING.

SEE ALSO

qdel(1), qdev(1), qlimit(1), qpr(1), and qsub(1).
qmgr(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

qsub - submit an NQS batch request

SYNOPSIS

qsub [*option* ...] [*script-file*]

DESCRIPTION

qsub submits a batch request to the Network Queuing System (NQS). If a *script-file* is not specified, *qsub* reads from **stdin**. All *script-files* are spooled so that later changes will not affect previously queued batch requests.

NQS has queue access restrictions. For each queue with a queue type other than "network", access may be either "unrestricted" or "restricted". If access is "unrestricted", any request may enter the queue. If access is "restricted", a request can only enter the queue if the requester or the requester's login group has access to that queue (see *qmgr*(1M)). Requests submitted by the super-user are an exception; they are always queued, even if the super-user has not explicitly been given access. *qstat*(1) may be used to determine who has access to a particular queue.

qpr prints a request-id to **stdout**, upon the successful queuing of a request. This request-id can be compared with what is reported by *qdev*(1) and *qstat*(1) to learn what happened to a request and given as an argument to *qdel*(1) to delete a request. A request-id has the form *seqno.host-name* where *seqno* refers to the sequence number assigned to the NQS request, and *host-name* refers to the name of originating machine. This identifier is used throughout NQS to uniquely identify the request anywhere in the network.

All of the command line *options* can also be specified within the first comment block in the batch request *script-file* as *embedded default* options. Such options appearing in the batch request *script-file* set default characteristics for the batch request. If the same option is specified on the command line, the command line option (and any associated value) takes precedence over the embedded option. The algorithm used to scan for embedded default options is as follows:

1. Read the first line of the *script-file*.
2. If the current line contains only white space characters, or the first nonwhite space character of the line is ":", go to step 7.
3. If the first nonwhite space character of the current line is not a "#", go to step 8.
4. If the second nonwhite space character in the current line is not "@" or the character immediately following the second nonwhite space character in the current line is not "$", go to step 7.
5. If a "-" is not the character immediately following the "@$" sequence, go to step 8.
6. Process the embedded option, stop the parsing process when the end of the line or the first unquoted "#" character is reached.

7. Read the next line of the *script-file*. Go to step 2.
8. End. Embedded options will no longer be recognized.

The following is an example of using embedded options withing a *script-file*.

```
            #
            # Batch request script example:
            #
            # @$-a "11:30pm EDT"
            #                    # Run request after 11:30 EDT by
default,
            # @$-mb -me          # Send mail at beginning and end of
            #                    # request execution.
            # @$-q batch1        # Submit request to queue, batch1 by
            #                    # default.
            # @$                 # No more embedded options.
            #
            make all
```

The following *options* are available.

-a *date-time*

Do not run the batch request before the specified date and/or time.

If a *date-time* specification is composed of two or more tokens separated by white space characters, the *date-time* specification must be in double quotes as in: "-a July, 4, 2026 12:31-EDT". If not specified in double quotes, it should be escaped so that the shell will interpret the *date-time* specification as a single lexical token.

The syntax accepted for the *date-time* parameter is flexible. Unspecified date and time values default to an appropriate value. (For example, if a date is not specified, the current month, day, and year are assumed.)

A date can be specified as a month and a day (current year assumed). The year can also be explicitly specified. It is also possible to specify the date as a weekday name (i.e., "Tues"), or as one of the strings "today" or "tomorrow." Weekday and month names can be abbreviated by any three-character (or longer) prefix to the actual name. An optional period can follow an abbreviated month or day name.

Time of day specifications can be given using a twenty-four hour clock or "am" and "pm" specifications may be used alternatively. When a meridian is not specified, a twenty-four hour clock is assumed.

The time of day specification is interpreted using the precise meridian definitions. "12am" refers to the twenty-four hour clock time of 0:00:00; "12m" refers to noon; and "12-pm" refers to 24:00:00. Alternatively, the phrases "midnight" and "noon" are accepted as time of day specifications, where "midnight" refers to 24:00:00.

A timezone may also appear at any point in the *date-time* specification. Thus, it is legal to say: "April 1, 1987 13:01-PDT". When a timezone is not specified, the local timezone is assumed, with daylight savings time being inferred when appropriate based on the date specified.

All alphabetic comparisons are not case-sensitive. Both "WeD" and "weD" refer to Wednesday.

Examples of valid *date-time* specifications are:

```
"01-Jan-1986 12am, PDT"
"Tuesday, 23:00:00"
"11pm tues."
"tomorrow 23-MST"
```

-mb Send mail to the user on the originating machine when the request begins execution. If the **-mu** option is also present, mail is sent to the user specified by the **-mu** option instead of to the invoking user.

-me Send mail to the invoking user on the originating machine when the request has ended execution. If the **-mu** option is also present, mail is sent to the user specified by the **-mu** option instead of to the invoking user.

-mu *user-name*
Specify that any mail concerning the request should be delivered to the user *user-name*. *User-name* has the form *user*[@*machine*]. When this option is not specified, any mail concerning the request is sent to the invoker on the originating machine.

-p *priority*
Assign an intraqueue priority to this request. The specified *priority* must be an integer, and must be in the range 0-63, inclusive. A value of 63 defines the highest intraqueue request priority, while a value of 0 defines the lowest. This priority does not determine the execution priority of the request. This priority is only used to determine the relative ordering of requests within a queue.

When a request is added to a queue, it is placed at a specific position within the queue so that it appears ahead of all existing requests with priorities less than the priority of the new request. Similarly, all requests with a higher priority remain ahead of the new request when the queuing process is complete. When the priority of the new request equals the priority of an existing request, the existing request takes precedence over the new request.

If the user does not choose an intraqueue priority, the value configured by the system administrator will be used. If a value has not been configured by the system administrator, a default value of 31 is assigned to the request.

-q *queue-name*

Specify the queue to which the batch request is to be submitted. If the **-q** *queue-name* specification is not given, the user's environment is searched for the variable QSUB_QUEUE. If this environment variable is found, the character string value for QSUB_QUEUE is presumed to be the name of the queue to which the request should be submitted. If the QSUB_QUEUE environment variable is not found, the request is submitted to the default batch request queue if one has been defined by the local system administrator. Otherwise, the request cannot be queued and an appropriate error message is displayed.

-r *request-name*

Assign a name to this request. When the **-r** option is not specified, the *request-name* defaults to *script-file* (leading path name removed) specified on the command line. If a *script-file* is not specified, the default *request-name* assigned to the request is **stdin**.

In all cases, if the *request-name* begins with a digit, the character "R" is prefixed to prevent a *request-name* from beginning with a digit. All *request-names* are truncated to a maximum length of 15 characters.

-e [*machine*:] *stderr-filename*

Direct **stderr** output generated by the batch request to *stderr-filename* on *machine*.

If an explicit *machine* destination is not specified, the destination machine defaults to the machine where the batch request originated or to the machine where the request will eventually be run, depending on the absence or presence of the **-ke** option.

If a *machine* destination is not specified and the *stderr-filename* does not begin with a "/", the current working directory is prefixed to create a fully-qualified path name, if the **-ke** option is absent. In all other cases, any partial *stderr-filename* is interpreted relative to the user's home directory on the **stderr** destination machine. This option cannot be specified when the **-eo** option is present.

If the **-eo** and **-e** options are not specified, all **stderr** output for the batch request is sent to the file whose name consists of the first seven characters of the request name followed by the characters ".e", followed by the sequence number portion of the request-id. Without the **-ke** option, the default **stderr** output file will be placed in the directory where the request was submitted on the originating machine. Otherwise, the file will be placed in the user's home directory on the execution machine.

-o [*machine*:] *stdout-filename*

Direct **stdout** output generated by the batch request to *stdout-filename* on *machine*.

If an explicit *machine* destination is not specified, the destination machine defaults to the machine where the batch request originated or to the machine where the request will eventually be run, depending on the absence or presence of the **-ko** option.

If a *machine* destination is not specified and the *stdout-filename* does not begin with a "/", the current working directory is prefixed to create a fully-qualified path name, if the **-ko** option is absent. In all other cases, any partial *stdout-filename* is interpreted relative to the user's home directory on the **stdout** destination machine.

If the **-o** option is not specified, all **stdout** output for the batch request is sent to the file whose name consists of the first seven characters of the request name followed by the characters ".o", followed by the sequence number portion of the request-id. Without the **-ko** option, the default **stdout** output file will be placed in the directory where the request was submitted on the originating machine. Otherwise, the file will be placed in the user's home directory on the execution machine.

-eo Direct all output, for the batch request, that would normally be sent to the **stderr** file to the **stdout** file. This option cannot be specified when the **-e** option is present.

-ke In the absence of an explicit *machine* destination for the **stderr** file produced by a batch request, the destination chosen is the machine where the batch request originated. The **-ke** option, however, instructs NQS to leave any **stderr** output file produced by the request on the machine where the batch request was executed.

This option is meaningless if the **-eo** option is specified and cannot be specified if an explicit *machine* destination is given with the **-e** option.

-ko In the absence of an explicit *machine* destination for the **stdout** file produced by a batch request, the destination chosen is the machine where the batch request originated. The **-ko** option, however, instructs NQS to leave any **stdout** output file produced by the request on the machine where the batch request was executed.

This option cannot be specified if an explicit *machine* destination is given with the **-o** option.

-lf *file-size-limit*

Set a per-process file size limit for all processes that constitute the running batch request. If any process in the running request attempts to write to a file such that the file size exceeds *file-size-limit*, that process is terminated by a signal chosen by the underlying implementation.

The format for *file-size-limit* is either *.fraction*[*units*] or *integer*[*.fraction*][*units*] when the limit is a finite limit. If an infinite limit is needed, the *file-size-limit* may be specified as "unlimited" or any initial substring. The *integer* and *fraction* portions of a finite limit may be specified as strings of up to eight decimal digits. The *units* may be specified as one of the following case-insensitive strings.

b	-bytes
w	-words
kb	-kilobytes (2^10 bytes)
kw	-kilowords (2^10 words)
mb	-megabytes (2^20 bytes)
mw	-megawords (2^20 words)
gb	-gigabytes (2^30 bytes)
gw	-gigabytes (2^30 words)

When *units* are not specified, bytes are assumed. If the limit is set to "unlimited", the only limitations imposed are those of the physical hardware involved.

-ln *nice-value*

Set a per-process nice value for all processes in the running batch request.

A nice value determines the execution-time priority of a process relative to all other processes in the system. By letting the user set a limit on the nice value for all processes in the running request, a user can cause a request to consume less (or more) of the CPU resources.

Increasingly negative nice values cause the relative execution priority of a process to increase, while increasingly positive nice values cause the relative priority to decrease. Thus, "-ln -10" has a higher execution priority than "-ln 0".

The *nice-value* must be acceptable to the batch queue in which the request is ultimately placed.

-nr NQS will, by default, restart, upon system boot, any request that were running at the time of an NQS shutdown or system crash. The **-nr** option will, however, not restart any request that were running. Requests that were not running are always preserved.

-s *shell-name*

Specify the absolute path name for the shell that will interpret the batch request script. This option unconditionally overrides any shell strategy configured on the execution machine. When this option is not specified, the NQS system on the execution machine will use one of three shell strategies, *fixed*, *free*, or *login* (see *qlimit*(1) for a description of the shell strategies), to determine the shell that will be used.

-x When a batch request is submitted, the current values of the following environment variables are automatically exported: HOME, SHELL, PATH, LOGNAME, MAIL, and TZ. When the batch request is spawned, these variables are re-created respectively as the environment variables QSUB_HOME, QSUB_SHELL, QSUB_PATH, QSUB_LOGNAME, QSUB_MAIL, and QSUB_TZ If the **-x** option is specified, all remaining environment variables, with names that do not conflict with the automatically-exported variables, are also exported. When the batch request is spawned, these additional variables are re-created under the same name.

-z Submit the batch request silently. If the request is submitted successfully, messages are not displayed indicating this fact. Error messages will, however, always be displayed.

The following sequence of events takes place when an NQS batch request is spawned.

1. The process that will head the process group for all processes composing the batch request is created by NQS.
2. Resource limits are enforced.
3. The real and effective group ID's of the process are set to the group ID as defined in the local password file of the request owner.
4. The real and effective user ID's of the process are set to the real user ID of the batch request owner.
5. The user file creation mask is set to the value that the user had on the originating machine when the batch request was submitted.
6. The shell with which to execute the batch request script is chosen.
7. The environment variables HOME, SHELL, PATH, LOGNAME, and MAIL are set from the user's password file entry as though the user had logged directly into the execution machine.
8. The environment string: ENVIRONMENT=BATCH is added to the environment so that shell scripts (and the user's **.profile** (Bourne shell) or **.cshrc** and **.login** (C-shell) scripts) can test for batch request execution when appropriate and not set any terminal characteristics, since a batch request is not connected to an input terminal.
9. The environment variables QSUB_WORKDIR, QSUB_HOST, QSUB_REQNAME, and QSUB_REQID are added to the environment. These environment variables equal the respective strings of the working directory when the request was submitted, the name of the originating host, the name of the request, and the id of the request.
10. All remaining environment variables saved for re-creation when the batch request is spawned are added at this point to the environment.
11. The current working directory is then set to the user's home directory on the execution machine and the chosen shell is exec'd. If the Bourne shell is chosen, the **.profile** is read. If the C-shell is chosen, the **.cshrc** and **.login** scripts are read. The batch request is then executed with the environment as constructed in the steps outlined above.

SEE ALSO

qdel(1), qdev(1), qlimit(1), qpr(1), qstat(1), setpgrp(2), signal(2).
qmgr(1M) in the *CLIX System Administrator's Reference Manual.*
kill(2) in the *UNIX System V Programmer Reference Manual.*
mail(1) in the *UNIX System V User's Reference Manual.*

NOTES

When an NQS batch request is spawned, a new process group is established so that all processes of the request exist in the same process group. If the *qdel*(1) command is used to send a signal to an NQS batch request, the signal is sent to all processes of the request in the created process group. However, if one or more processes of the request successfully executes a *setpgrp*(2) system call, the processes will not receive signals sent by the *qdel*(1) command. The *kill*(1) command may be used to delete such processes.

All processes of an NQS request should catch SIGTERM signals. By default, the receipt of a SIGTERM signal causes the receiving process to die. NQS sends a SIGTERM signal to all processes in the established process group for a batch request as a notification that the request should be prepared to be killed.

The spawned shell ignores SIGTERM signals. If the current immediate child of the shell does not ignore or catch SIGTERM signals, it will be killed by the receipt of such and the shell will proceed to execute the next command from the script (if there is one). In any case, the shell will not be killed by the SIGTERM signal, though the executing command will have been killed.

After receiving a SIGTERM signal delivered from NQS, a process of a batch request typically has 60 seconds before receiving a SIGKILL signal (the 60-second duration can be changed by the operator).

A sufficient method to echo commands executed by unmodified versions of the Bourne shell and C-shell are not available. While the C-shell can be spawned so that it echoes the commands it executes, it is often difficult to distinguish an echoed command from output produced by the batch request because a magic character such as a "$" is not displayed in front of the echoed command. The Bourne shell does not support any echo option. Thus, one of the better ways to write the shell script for a batch request is to place lines in the shell script of the form:

```
echo "explanatory-message"
```

CAVEATS

Network queues have not yet been implemented.

In this implementation, it is not possible to see the **stderr** or **stdout** files produced by the batch request while the request is running unless the **-re** and **-ro** options have been respectively specified.

NAME

ratfor - rational FORTRAN dialect

SYNOPSIS

ratfor [**-h**] [**-C**] [**-6***x*] [*file* ...]

DESCRIPTION

ratfor converts a rational dialect of FORTRAN into ordinary irrational FORTRAN. Ratfor provides control flow constructs essentially identical to those in C:

statement grouping:	{ *statement*; *statement*; *statement* }
decision-making:	**if** (*condition*) *statement* [**else** *statement*]
	switch (*integer-expression*) {
	case *integer-expression*: *statement*
	...
	[**default:**] *statement*
	}
loops:	**while** (*condition*) *statement*
	for (*expression*; *condition*; *expression*) *statement*
	do *limits statement*
	repeat *statement* [**until** (*condition*)]
	break
	next

and some syntactic sugar to make programs easier to read and write:

free form input:	; - multiple statements/line
	automatic continuation
comments:	# this is a comment.
translation of relationals:	
	>, >=, etc., become .GT., .GE., etc.
return expression to caller from function:	
	return (*expression*)
define:	**define** *name replacement*
include:	**include** *file*

The option **-h** causes quoted strings to be turned into 27H constructs. The **-C** option copies comments to the output and attempts to format it neatly. Normally, continuation lines are marked with a **&** in column 1; the option **-6***x* makes the continuation character *x* and places it in column 6.

ratfor is best used with *f77*(1).

SEE ALSO

efl(1), f77(1).

NAME

rcmd - remote command

SYNOPSIS

rcmd *host* [-l *user-name*] [**-n**] [*command*]

DESCRIPTION

rcmd connects to the specified *host* and executes the specified *command*. *rcmd* copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit, and terminate signals are propagated to the remote command; *rcmd* normally terminates when the remote command does.

The remote user name used is the same as the local user name, unless a different remote name is specified with the -l option. This remote name must be equivalent to the originating account; no provision is made for specifying a password with a command. If no input to the remote command is desired, the **-n** option is used to redirect **stdin** of *rcmd* to **/dev/null**.

If *command* is omitted, instead of executing a single command, the remote host will be logged in to using *rlogin*(1).

Unquoted shell metacharacters are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command

rcmd otherhost cat *remote-file* >> *local-file*

appends the remote file *remote-file* to the local file *local-file*, while

rcmd otherhost cat *remote-file* ">>" *other-remote-file*

appends the remote file *remote-file* to the other remote file *other-remote-file*.

Host names are given in the file **/etc/hosts**. Each host has one standard name (the first name given in the file), which is long and unambiguous and optionally one or more nicknames.

FILES

/etc/hosts

SEE ALSO

rlogin(1).

BUGS

An interactive command (like *vi*(1)) cannot be run and *rlogin*(1) cannot be used.

"Stop" signals stop the local *rcmd* process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

NAME

rcp - remote file copy

SYNOPSIS

rcp [**-p**] *file1 file2*
rcp [**-p**] [**-r**] *file* ... *directory*

DESCRIPTION

rcp copies files between machines. Each *file* or *directory* argument is either a remote file name of the form *rhost:path* or a local file name (either containing no ":" characters or having a "\" before any ":").

If the **-r** option is specified and any of the source files are directories, *rcp* copies each subtree rooted at that name. In this case, the destination must be a directory.

By default, the mode and owner of *file2* are preserved if it already existed. Otherwise, the mode of the source file modified by the *umask*(1) on the destination host is used. The **-p** option attempts to preserve (duplicate) in the copies the modification times and modes of the source files, ignoring the *umask*(1).

If *file* is not a full path name, it is interpreted relative to the login directory on the remote host. A *file* on a remote host may be quoted (using \, ", or ') so that the metacharacters are interpreted remotely.

rcp does not prompt for passwords. The remote user name is assumed to be the same as the local user name unless the remote file argument has the form *ruser@rhost:path*.

SEE ALSO

ftp(1), rcmd(1), rlogin(1).
cp(1), umask(1) in the *UNIX System V User's Reference Manual*.

BUGS

rcp does not detect all cases where the target of a copy might be a file when only a directory should be legal.

rcp is confused by any output generated by commands in a **.login**, **.profile**, or **.cshrc** file on the remote host.

NAME

rcs - change RCS file attributes

SYNOPSIS

rcs [*option* ...] *file* ...

DESCRIPTION

rcs creates new Revision Control System (RCS) files or changes attributes of existing ones. An RCS *file* contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For *rcs* to work, the caller's login name must be on the access list unless the access list is empty, the caller is the owner of the *file* or the super-user, or the -i option is present.

Files ending in ",v" are RCS files. All others are working files. If a working file is given, *rcs* tries to find the corresponding RCS file first in directory **./RCS** and then in the current directory as explained in *co*(1).

-i	Creates and initializes a new RCS *file*, but does not deposit any revision. If the RCS *file* has no path prefix, *rcs* tries to place it first in the subdirectory **./RCS** and then in the current directory. If the RCS *file* exists, an error message is printed.
-a*logins*	Appends the login names appearing in the comma-separated list *logins* to the access list of the RCS *file*.
-A*oldfile*	Appends the access list of *oldfile* to the access list of the RCS *file*.
-e[*logins*]	Erases the login names appearing in the comma-separated list *logins* from the access list of the RCS *file*. If *logins* is omitted, the entire access list is erased.
-b[*rev*]	Sets the default branch to *rev*. If *rev* is omitted, the default branch is reset to the (dynamically) highest branch on the trunk.
-c*string*	Sets the comment leader to *string*. The comment leader is printed before every log message line generated by the keyword **$Log$** during checkout (see *co*(1)). This is useful for programming languages without multiline comments. During initial checkin, the comment leader is determined from the suffix of the working *file*.
-l[*rev*]	Locks the revision with number *rev*. If a branch is given, the latest revision on that branch is locked. If *rev* is omitted, the latest revision on the default branch is locked. Locking prevents overlapping changes. A lock is removed with *ci*(1) or **rcs -u** (see below).
-u[*rev*]	Unlocks the revision with number *rev*. If a branch is given, the latest revision on that branch is unlocked. If *rev* is

omitted, the most recent lock held by the caller is removed. Otherwise, the most recent lock is broken. Normally, only the locker of a revision may unlock it. Somebody else unlocking a revision breaks the lock. This sends a mail message to the original locker. The message contains a commentary solicited from the breaker. The commentary is terminated with a line containing a single "." or <CONTROL>-D.

-L	Sets locking to strict. Strict locking means that the owner of an RCS file is not exempt from locking for checkin. This option should be used for shared files.
-U	Sets locking to nonstrict. Nonstrict locking means that the owner of a file need not lock a revision for checkin. This option should not be used for shared files. The default (**-L** or **-U**) is determined by the system administrator.
-n*name*[:*rev*]	Associates the symbolic name *name* with the branch or revision *rev*. *rcs* prints an error message if *name* is already associated with another number. If *rev* is omitted, the symbolic name is deleted.
-N*name*[:*rev*]	Same as **-n**, except that it overrides a previous assignment of *name*.
-o*range*	Deletes ("outdates") the revisions given by *range*. A range consisting of a single revision number denotes that revision. A range consisting of a branch number denotes the latest revision on that branch. A range with the form *rev1-rev2* represents revisions *rev1* to *rev2* on the same branch, *-rev* represents the beginning of the branch containing *rev* up to and including *rev*, and *rev-* represents revision *rev* to the end of the branch containing *rev*. None of the outdated revisions may have branches or locks.
-q	Quiet mode. Diagnostics are not printed.
-s*state*[:*rev*]	Sets the state attribute of the revision *rev* to *state*. If *rev* is a branch number, the latest revision on that branch is assumed. If *rev* is omitted, the latest revision on the default branch is assumed. Any identifier is acceptable for *state*. A useful set of states is **Exp** (for experimental), **Stab** (for stable), and **Rel** (for released). By default, *ci*(1) sets the state of a revision to **Exp**.
-t[*txtfile*]	Writes descriptive text to the RCS *file*. (Deletes the existing text.) If *txtfile* is omitted, *rcs* prompts the user for text supplied from the standard input, terminated with a line containing a single "." or <CONTROL>-D. Otherwise, the descriptive text is copied from the file *txtfile*. If the **-i** option is present, descriptive text is requested even if **-t** is

not given. The prompt is suppressed if the standard input is not a terminal.

FILES

The caller of the command must have read/write permission for the directory containing the RCS *file* and read permission for the RCS *file* itself. *rcs* creates a semaphore file in the same directory as the RCS *file* to prevent simultaneous update. For changes, *rcs* always creates a new file. On successful completion, *rcs* deletes the old one and renames the new one. This strategy makes links to RCS files useless.

SEE ALSO

co(1), ci(1), ident(1), rcsclean(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(4), sccstorcs(1).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

DIAGNOSTICS

The RCS *file* name and the revisions outdated are written to the diagnostic output. The exit status always refers to the last RCS *file* operated on, and is 0 if the operation was successful and 1 otherwise.

IDENTIFICATION

Author: Walter F. Tichy,
Purdue University, West Lafayette, IN 47907.
Copyright © 1982 by Walter F. Tichy.

NAME

rcsclean - clean up working files

SYNOPSIS

rcsclean [-r*rev*] [**-q**] *file* ...

DESCRIPTION

rcsclean removes working files that were checked out and never modified. For each *file* given, *rcsclean* compares the working *file* and a revision in the corresponding Revision Control System (RCS) *file*. If it finds no difference, it removes the working *file*, and, if the revision was locked by the caller, unlocks the revision.

A file name ending in ",v" is an RCS *file* name. Otherwise, it is a working *file* name. *rcsclean* derives the working *file* name from the RCS *file* name and vice versa, as explained in *co*(1). Pairs consisting of both an RCS and a working *file* name may also be specified.

Rev specifies the revision the working *file* is compared to. If *rev* is omitted, *rcsclean* compares the working *file* to the latest revision on the default branch (normally the highest branch on the trunk). The option **-q** suppresses diagnostics.

rcsclean is useful for "clean" targets in makefiles. Note that *rcsdiff*(1) prints the differences. Also, *ci*(1) normally asks whether to check in a *file* if it was not changed.

EXAMPLES

The command

```
rcsclean *.c *.h
```

removes all working files ending in ".c" or ".h" that were not changed since their checkout.

SEE ALSO

co(1), ci(1), ident(1), rcs(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(4).

DIAGNOSTICS

The exit status is 0 if there were no differences during the last comparison or if the last working file did not exist, 1 if there were differences, and 2 if there were errors.

IDENTIFICATION

Author: Walter F. Tichy,
Purdue University, West Lafayette, IN 47907.
Copyright © 1982 by Walter F. Tichy.

NAME

rcsdiff - compare RCS revisions

SYNOPSIS

rcsdiff [**-b**] [**-cefhn**] [**-q**] [-r*rev1*] [-r*rev2*] *file* ...

DESCRIPTION

rcsdiff compares two revisions of each Revision Control System (RCS) *file* given. A *file* name ending in ",v" is an RCS file name; otherwise, it is a working *file* name. *rcsdiff* derives the working *file* name from the RCS *file* name and vice versa, as explained in *co*(1). Pairs consisting of an RCS and a working *file* name may also be specified.

Except for **-b**, **-q** and **-r**, which may be used with any other options, the following options are mutually exclusive.

-e Produces a script of **a**, **c** and **d** commands for the editor *ed*(1), which will recreate *file2* from *file1*. In connection with **-e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed*(1) scripts ($2, $3, ...) made by *diff*(1) need be available. A "latest version" appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Extra commands are added to the output when comparing directories with **-e** so that the result is a *sh*(1) script for converting text files common to the two directories from their state in *dir1* to their state in *dir2*.

-f Produces a script similar to that of **-e**, not useful with *ed*(1), and in the opposite order.

-c[#] Produces a *diff*(1) with lines of context. The default is to present 3 lines of context and may be changed (for example to 10) by **-c10**. With **-c**, the output format is modified slightly: the output begins by identifying files involved and their creation dates, and then each change is separated by a line with 12 *'s. The lines removed from *file1* are marked with "— "; those added to *file2* are marked "+ ". Lines changed from one file to the other are marked in both files with "! ".

Changes that are in context lines of each other are grouped on output. (This is a change from the previous "diff -c", but the resulting output is usually much easier to interpret.)

-h Does a fast, less thorough job. It works only when changed stretches are short and well separated, but does work on *files* with unlimited length.

-n Generates an edit script of the format used by RCS.

-q Suppresses diagnostic output.

-b Causes trailing blanks (spaces and tabs) to be ignored, and other strings of blanks to compare equally.

-r If *rev1* and *rev2* are omitted, *rcsdiff* compares the latest revision on the default branch (normally the highest branch on the trunk) with the contents of the corresponding working *file*. This is useful for determining what was changed since the last checkin.

If *rev1* is given, but *rev2* is omitted, *rcsdiff* compares revision *rev1* of the RCS *file* with the contents of the corresponding working *file*.

If *rev1* and *rev2* are given, *rcsdiff* compares revisions *rev1* and *rev2* of the RCS *file*.

Rev1 and *rev2* may be given numerically or symbolically, and may actually be attached to any of the options.

EXAMPLES

The command

```
rcsdiff f.c
```

produces differences on the latest revision on the default branch of RCS file **f.c,v** and the contents of working file **f.c**.

SEE ALSO

ci(1), co(1), ident(1), rcs(1), rcsclean(1), rcsmerge(1), rlog(1), rcsfile(4).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software*, IEEE, Tokyo, Sept. 1982.

DIAGNOSTICS

The exit status is 0 if there were no differences during the last comparison, 1 if there were differences, and 2 if there were errors.

IDENTIFICATION

Author: Walter F. Tichy,
Purdue University, West Lafayette, IN 47907.
Copyright © 1982 by Walter F. Tichy.

NAME

rcsmerge - merge RCS revisions

SYNOPSIS

rcsmerge -r*rev1* [-r*rev2*] [-**p**] *file*

DESCRIPTION

rcsmerge incorporates the changes between *rev1* and *rev2* of a Revision Control System (RCS) file into the corresponding working file. If **-p** is given, the result is printed on the standard output. Otherwise, the result overwrites the working file.

A file name ending in ",v" is an RCS file name; otherwise, it is a working file name. *rcsmerge* derives the working file name from the RCS file name and vice versa, as explained in *co*(1). A pair consisting of an RCS and a working file name may also be specified.

Rev1 may not be omitted. If *rev2* is omitted, the latest revision on the default branch (normally the highest branch on the trunk) is assumed. *Rev1* and *rev2* may be given numerically or symbolically.

rcsmerge prints a warning if there are overlaps and delimits the overlapping regions as explained in the *co*(1) **-j** option. The command is useful for incorporating changes into a checked-out revision.

EXAMPLES

Suppose revision 2.8 of **f.c** has just been released. Revision 3.4 has just been completed when updates to release 2.8 are received from someone else. To combine the updates to 2.8 and the changes between 2.8 and 3.4, put the updates to 2.8 in file **f.c** and execute the following:

```
rcsmerge -p -r2.8 -r3.4 f.c >f.merged.c
```

Then, examine **f.merged.c**. Alternatively, to save the updates to 2.8 in the RCS file, check them in as revision 2.8.1.1 and execute **co -j**:

```
ci -r2.8.1.1 f.c
co -r3.4 -j2.8:2.8.1.1 f.c
```

As another example, the following command undoes the changes between revision 2.4 and 2.8 in the currently-checked-out revision in **f.c**.

```
rcsmerge -r2.8 -r2.4 f.c
```

Note the order of the arguments and that f.c will be overwritten.

SEE ALSO

ci(1), co(1), merge(1), ident(1), rcs(1), rcsclean(1), rcsdiff(1), rlog(1), rcsfile(4).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

BUGS

rcsmerge does not work with files that contain lines with a single ".".

IDENTIFICATION

Author: Walter F. Tichy,
Purdue University, West Lafayette, IN 47907.
Copyright © 1982 by Walter F. Tichy.

NAME

restore - incremental file system restore

SYNOPSIS

/etc/restore *key* [*name* ...]

DESCRIPTION

restore reads tapes backed up with the *backup*(1) command. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying the files to be restored. Unless the **h** *key* is specified (see below), the appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the *key* is specified by one of the following letters:

r The tape is read and loaded in the current directory. This should not be done lightly; the **r** *key* should only be used to restore a complete backup tape on a clear file system or to restore an incremental backup tape after a full-level 0 restore. Thus

```
mkfs 3000 /dev/dsk/s0u0p7.4
mount /dev/dsk/s0u0p7.4 /mnt
cd /mnt
restore r
```

is a typical sequence to restore a complete backup. Another *restore* can be done to put an incremental backup on top of this. Note that *restore* leaves a file **restoresymtab** in the root directory to pass information between incremental restore passes. This file should be removed when the last incremental tape has been restored.

A *backup*(1) followed by a *mkfs*(1) and a *restore* is used to change the size of a file system.

R *restore* requests a particular tape of a multivolume set to restart a full restore (see the **r** *key* above). This allows *restore* to be interrupted and then restarted.

x The named files are extracted from the tape. If the named file matches a directory whose contents were written to the tape and the **h** *key* is not specified, the directory is recursively extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the root directory is extracted, which results in the entire contents of the tape being extracted unless the **h** *key* was specified.

t The names of the specified files are listed if they occur on the tape. If no file argument is given, the root directory is listed, which results in the entire contents of the tape being listed unless the **h** *key* has been specified.

i This mode allows files to be interactively restored from a backup tape. After reading the directory information from the tape, *restore* provides a shell-like interface that allows the user to move around the directory tree, selecting files to be extracted. The available commands are given below; for the commands that require an argument, the default is the current directory.

ls [*dir*] List the current or specified directory. Entries that are directories are appended with a "/". Entries that have been marked for extraction are prepended with a "*". If the verbose *key* is set, the i-node number of each entry is also listed.

cd *dir* Change the current working directory to the specified argument.

pwd Print the full path name of the current working directory.

add [*arg*] The current directory or specified argument is added to the list of files to be extracted. If a directory is specified, it and all its descendants are added to the extraction list (unless the **h** *key* is specified on the command line). Files on the extraction list are prepended with a "*" when they are listed by *ls*(1).

delete [*arg*] The current directory or specified argument is deleted from the list of files to be extracted. If a directory is specified, it and all its descendants are deleted from the extraction list (unless the **h** *key* is specified on the command line). The most expedient way to extract most files from a directory is to add the directory to the extraction list and then delete the files that are not needed.

extract All files on the extraction list are extracted from the backup tape. *restore* will ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume and work toward the first volume.

setmodes All directories added to the extraction list have their owner, modes, and times set; nothing is extracted from the tape. This is useful for cleaning up after a restore has been prematurely aborted.

verbose The sense of the **v** *key* is toggled. When set, the verbose *key* causes the **ls** command to list the i-node numbers of all entries. It also causes *restore* to print information about each file as it is extracted.

help List a summary of the available commands.

quit *restore* immediately exits even if the extraction list is not empty.

The following characters may be used in addition to the letter that selects the function desired.

b The next argument to *restore* is used as the block size of the tape (in kilobytes). If the **-b** option is not specified, *restore* tries to determine the tape block size dynamically.

f The next argument to *restore* is used as the name of the archive instead of **/dev/rmt/0m**. *rtc*(1) can be used to restore tapes from a remote tape device. If the name of the file is "-", *restore* reads from standard input. Thus, *backup*(1) and *restore* can be used in a pipeline to backup and restore a file system with the following command:

```
backup 0f - /usr | (cd /mnt; restore xf -)
```

v Normally *restore* works silently. The **v** (verbose) *key* causes it to type the name of each file it treats preceded by its file type.

y *restore* will not ask whether it should abort the restore if it gets a tape error. It will always try to skip the bad tape block(s) and continue.

m *restore* will extract by i-node numbers rather than by file name. This is useful if only a few files are being extracted to avoid regenerating the complete path name to the file.

h *restore* extracts the actual directory, rather than the files that it references. This prevents hierarchical restoration of complete subtrees from the tape.

s The next argument to *restore* is a number that selects the file on a multifile backup tape. File numbering starts at 1.

FILES

/dev/rmt/0m	the default tape drive
/tmp/rstdir*	file containing directories on the tape
/tmp/rstmode*	owner, mode, and time stamps for directories
./restoresymtab	information passed between incremental restores

SEE ALSO

backup(1), rtc(1).
newfs(1M), mount(1M) in the *CLIX System Administrator's Reference Manual*.
mkfs(1M) in the *UNIX System Administrator's Reference Manual*.

DIAGNOSTICS

Complains about bad *key* characters.

Complains if it gets a read error. If **y** has been specified or the user responds "y", *restore* will attempt to continue the restore.

If the backup extends over more than one tape, *restore* will ask the user to change tapes. If the **x** or **i** *key* has been specified, *restore* will also ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume and work toward the first volume.

Numerous consistency checks can be listed by *restore*. Most checks are self-explanatory or can "never happen". Common errors are given below.

File-name: not found on tape
: The specified *file-name* was listed in the tape directory, but was not found on the tape. This is caused by tape read errors while looking for the file and from using a backup tape created on an active file system.

Expected next file *inumber*, got *inumber*
: A file that was not listed in the directory appeared. This can occur when using a backup tape created on an active file system.

Incremental tape too low
: When performing incremental restore, a tape that was written before the previous incremental tape or that has an incremental level that is too low was loaded.

Incremental tape too high
: When performing incremental restore, a tape that does not begin its coverage where the previous incremental tape left off, or that has an incremental level that is too high was loaded.

Tape read error while restoring *file-name*
Tape read error while skipping over i-node *inumber*
Tape read error while trying to resynchronize
: A tape read error occurred. If a file name is specified, its contents are probably partially wrong. If an i-node is being skipped or the tape is trying to resynchronize, no extracted files have been corrupted, though files may not be found on the tape.

Resync restore, skipped *num* blocks
: After a tape read error, *restore* may need resynchronize. This message lists the number of blocks that were skipped.

BUGS

restore can become confused when performing incremental restores from backup tapes made on active file systems.

A level zero backup must be performed after a full *restore*. Because *restore* runs in user code, it cannot control i-node allocation; thus, a full *restore* must be performed to get a new set of directories. These directories reflect the new i-node numbering even though the file contents are unchanged.

NAME

rlog - print log messages and other information about RCS files

SYNOPSIS

rlog [*option* ...] *file* ...

DESCRIPTION

rlog prints information about Revision Control System (RCS) files. Files ending in ",v" are RCS files; all others are working files. If a working file is given, *rlog* searches for the corresponding RCS file first in directory **./RCS** and then in the current directory, as explained in *co*(1).

rlog prints the following information for each RCS file: RCS file name, working file name, head (i.e., the number of the latest revision on the trunk), default branch, access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for printing, and descriptive text. This is followed by entries for the selected revisions in reverse chronological order for each branch. For each revision, *rlog* prints revision number, author, date/time, state, number of lines added/deleted (with respect to the previous revision), locker of the revision (if any), and log message. Without options, *rlog* prints complete information. The options below restrict this output.

-L Ignores RCS files that have no locks set; convenient when combined with **-R**, **-h**, or **-l**.

-R Prints only the RCS file name; convenient for translating a working file name into an RCS file name.

-h Prints only the RCS file name, working file name, head, default branch, access list, locks, symbolic names, and suffix.

-t Prints the same as **-h**, plus the descriptive text.

-b Prints information about the revisions on the default branch (normally the highest branch on the trunk).

-d*dates* Prints information about revisions with a checkin date/time in the ranges given by the semicolon-separated list of *dates*. A range with the form *d1*<*d2* or *d2*>*d1* selects the revisions deposited between *d1* and *d2* (inclusive). A range with the form <*d* or *d*> selects all revisions dated *d* or earlier. A range with the form *d*< or >*d* selects all revisions dated *d* or later. A range with the form *d* selects the single, latest revision dated *d* or earlier. The date/time strings *d*, *d1*, and *d2* are in the free format explained in *co*(1). Quoting is normally necessary, especially for < and >. Note that the separator is a semicolon.

-l[*lockers*] Prints information about locked revisions. If the comma-separated list *lockers* of login names is given, only the revisions locked by the given login names are printed. If the list is omitted, all locked revisions are printed.

-r*revisions* Prints information about revisions given in the comma-separated list *revisions* of revisions and ranges. A range *rev1-rev2* indicates revisions *rev1* to *rev2* on the same branch; *-rev* indicates revisions from the beginning of the branch up to and including *rev*; and *rev-* indicates revisions starting with *rev* to the end of the branch containing *rev*. An argument that is a branch indicates all revisions on that branch. A range of branches indicates all revisions on the branches in that range.

-s*states* Prints information about revisions whose state attributes match one of the states given in the comma-separated list *states*.

-w[*logins*] Prints information about revisions checked in by users with login names appearing in the comma-separated list *logins*. If *logins* is omitted, the user's login is assumed.

rlog prints the intersection of the revisions selected with options **-d**, **-l**, **-s**, and **-w**, intersected with the union of the revisions selected by **-b** and **-r**.

EXAMPLES

```
rlog -L -R RCS/*,v
rlog -L -h RCS/*,v
rlog -L -l RCS/*,v
rlog RCS/*,v
```

The first command prints the names of all RCS files that have locks in the subdirectory **RCS**. The second command prints the headers of those files, and the third prints the headers plus the log messages of the locked revisions. The last command prints complete information.

SEE ALSO

ci(1), co(1), ident(1), rcs(1), rcsclean(1), rcsdiff(1), rcsmerge(1), rcsfile(4), sccstorcs(1).

Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

DIAGNOSTICS

The exit status always refers to the last RCS file operated on and is 0 if the operation was successful, 1 otherwise.

IDENTIFICATION

Author: Walter F. Tichy,
Purdue University, West Lafayette, IN 47907.
Copyright © 1982 by Walter F. Tichy.

NAME

rlogin - remote login

SYNOPSIS

rlogin *rhost* [**-e***c*] [**-l** *user-name*]

rhost [**-e***c*] [**-l** *user-name*]

DESCRIPTION

rlogin connects the terminal on the local host to the remote host *rhost*.

The remote terminal type is the same as the local terminal type (as given in the environment variable **TERM**). All echoing occurs at the remote host, so that (except for delays) the *rlogin* is transparent. Flow control through <CONTROL>-S and <CONTROL>-Q is handled on the remote host. A **~.** will disconnect the local host from the remote host, where ~ is the escape character. A **~<CONTROL>-Z**, where <CONTROL>-Z is the suspend character, will suspend the *rlogin* session.

The following options are supported:

-e*c*	Specify an escape character, *c*, other than ~.
-l *user-name*	Specify *user-name* to be used in the login procedure.

The command shown on the second line of the synopsis will allow the user to specify only the name of the remote host to be connected to rather than explicitly typing **rlogin**. To use this command, the user must link (see *ln*(1)) **/usr/bin/rlogin** to **/usr/bin/***rhost*, where *rhost* is the name of the remote host system.

If **/etc/hosts.equiv** exists, it contains a list of remote host names that share account names with the local host. (The host names must be the standard names as described in *rcmd*(1).) Users may also have a private equivalence list in a **.rhosts** file in their login directory. Each line in this file should contain an *rhost* and a *user-name* separated by a space, giving additional cases where logins without passwords will be permitted. If the originating user is not equivalent to the remote user, a login and password will be prompted for on the remote machine as in *login*(1). Either the remote user or root must own the **.rhosts** file.

SEE ALSO

rcmd(1).

NAME

rm, rmdir - remove files or directories

SYNOPSIS

rm [**-f**] [**-i**] *file* ...
rm -r [**-f**] [**-i**] *dirname* ... [*file* ...]
rmdir [**-p**] [**-s**] *dirname* ...

DESCRIPTION

rm removes the entries for one or more files from a directory. If an entry is the last link to the file, the file is destroyed. If an entry is a symbolic link, removal of the link file produces the same result as removal of an ordinary ("hard") link. However, if an entry is the file to which a symbolic link points, the file is removed, but the link remains and points to nothing. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed, followed by a question mark. This is a prompt for confirmation. If the answer begins with **y** (for "yes"), the file is deleted; otherwise the file remains.

Note that if the standard input is not a terminal, the command will operate as if the **-f** option is in effect.

rmdir removes the named directories, which must be empty.

Three options apply to *rm*:

-f This option removes all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (regardless of their permissions), but no messages are displayed. If the removal of a write-protected directory was attempted, this option could not suppress an error message.

-r This option causes any directories and subdirectories in the argument list to be recursively removed. Files will be emptied from the directory and the directory will be removed. Note that the user is normally prompted for removal of any write-protected files the directory contains. The write-protected files are removed without prompting, however, if the **-f** option is used or if the standard input is not a terminal and the **-i** option is not used.

If the removal of a nonempty, write-protected directory is attempted, the command will always fail (even if the **-f** option is used), resulting in an error message.

-i With this option, removal of any write-protected file is confirmed interactively. It overrides the **-f** option and remains in effect even if the standard input is not a terminal.

Two options apply to *rmdir*:

-p This option allows users to remove the directory *dirname* and its parent directories that become empty. A message is printed on standard output telling whether the whole path is removed or part of the path remains for some reason.

-s This option is used to suppress the message printed on standard error when **-p** is in effect.

SEE ALSO

ln(1).
unlink(2), rmdir(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

All messages are generally self-explanatory.

Removing the files "." and ".." is forbidden to avoid the consequences of inadvertently making the following type of mistake:

```
rm -r .*
```

Both *rm* and *rmdir* return exit codes of 0 if all specified directories are removed successfully. Otherwise, they return a nonzero exit code.

NAME

rpcgen - an RPC protocol compiler

SYNOPSIS

rpcgen -h [**-o** *outfile*] [*infile*]
rpcgen -c [**-o** *outfile*] [*infile*]
rpcgen *infile*
rpcgen [**-s** *transport*] [**-o** *outfile*] [*infile*]

DESCRIPTION

rpcgen is a tool that generates C code to implement a Remote Procedure Call (RPC) protocol. The input to *rpcgen* is a language with striking similarity to C known as the Remote Procedure Call Language (RPCL). *rpcgen* operates in four modes. The first mode is used to convert RPCL definitions into C definitions for use as a header file. The second mode compiles the External Data Representation (XDR) routines required to serialize the protocol described by RPCL. The third mode compiles both, leaving the header file in a file named *infile* with a extension. The fourth mode is used to compile an RPC server skeleton, so that an RPC server can be implemented using local procedures that know nothing about RPC.

The input may contain C-style comments and preprocessor directives. Comments are ignored, while the directives are simply written uninterpreted in the output header file.

Some of the XDR routines can be customized by leaving certain data types undefined. For every data type that is undefined, *rpcgen* assumes that there exists a routine with the name "xdr_" prepended to the name of the undefined type.

The following options are available:

-c	Compile XDR routines.
-h	Compile C data-definitions (a header file)
-o *outfile*	Specify the name of the output file. If none is specified, standard output is used (**-c**, **-h** and **-s** modes only).
-s *transport*	Compile a server, using the given transport. The supported transports are **udp** and **tcp**. This option may be invoked more than once so as to compile a server that serves multiple transports.

The following summary of RPCL syntax, which is used for *rpcgen* input, is intended more for aiding comprehension than as an exact statement of the language.

Primitive Data Types

[**unsigned**] **char**	[**unsigned**] **short**
[**unsigned**] **int**	[**unsigned**] **long**
unsigned	**float**
double	**void**

bool

Except for the added boolean data-type **bool**, RPCL is identical to C. *rpcgen* converts **bool** declarations to **int** declarations in the output header file (literally it is converted to a **bool_t**, which has been **#define**'d to be an **int**). Also, **void** declarations may appear only inside of **union** and **program** definitions. For those averse to typing the prefix **unsigned**, the abbreviations **u_char**, **u_short**, **u_int** and **u_long** are available.

Declarations

RPCL allows only three kinds of declarations:

declaration:
- *simple-declaration*
- *pointer-declaration*
- *vector-declaration*

simple-declaration:
type-name object-ident

pointer-declaration:
*type-name *object-ident*

vector-declaration:
type-name object-ident[*size*]

(*Size* can be either an integer or a symbolic constant).

RPCL declarations contain both limitations and extensions with respect to C. The limitations are that multidimensional arrays pointers-to-pointers cannot be declared in-line (they may still be declared using **typedef**). There are two extensions:

Opaque data is declared as a vector as follows:

opaque *object-ident* [*size*]

In the protocol, this results in an object of *size* bytes. Note that this is *not* the same as a declaration of *size* characters, since XDR characters are 32-bits. Opaque declarations are compiled in the output header file into character array declarations of *size* bytes.

Strings are special and are declared as a vector declaration:

string *object-ident* [*max-size*]

If *max-size* is unspecified, then there is essentially no limit to the maximum length of the string. String declarations get compiled into the following:

char **object-ident*

Type Definitions

The only way to generate an XDR routine is to define a type. For every type *zetype* defined, there is a corresponding XDR routine named *xdr_zetype*.

There are six ways to define a type:

type-definition:
  typedef
  enumeration-def
  structure-def
  variable-length-array-def
  discriminated-union-def
  program-def

The first three are very similar to their C namesakes. C does not have a formal type mechanism to define variable-length arrays and XDR unions are quite different from their C counterparts. Program definitions are not really type definitions, but they are useful nonetheless.

XDR definitions may not be nested. For example, the following will cause *rpcgen* to fail:

```
        struct dontdoit {
                struct ididit {
                        int oops;
                } sorry;
                enum ididitagain { OOPS, WHOOPS } iapologize;
        };
```

Typedefs

An XDR **typedef** looks as follows:

typedef:
  typedef *declaration* **;**

The *object-ident* part of *declaration* is the name of the new type, whereas the *type-name* part is the name of the type from which it is derived.

Enumeration Types

The syntax is:

enumeration-def:
  enum *enum-ident* **{**
    enum-list
  };

enum-list:
  enum-symbol-ident **[** = *assignment* **]**
  enum-symbol-ident **[** = *assignment* **]** **,** *enum-list*

(*assignment* may be either an integer or a symbolic constant)

If there is no explicit assignment, then the implicit assignment is the value of the previous enumeration plus 1. If not explicitly assigned, the first enumeration receives the value 0.

Structures

structure-def:
  struct *struct-ident* **{**
    declaration-list
  };

declaration-list:
 declaration ;
 declaration ; *declaration-list*

Variable-Length Arrays

variable-length-array-def:
```
        array array-ident {
                unsigned length-identifier ;
                vector-declaration ;
        };
```

A variable length array definition looks much like a structure definition. Here's an example:
```
        array mp_int {
                unsigned len;
                short val[MAX_MP_LENGTH];
        };
```

This is compiled into:
```
        struct mp_int {
                unsigned len;
                short *val;
        };
        typedef struct mp_int mp_int;
```

Discriminated Unions

discriminated-union-def:
```
        union union-ident switch ( discriminant-declaration ) {
                case-list
                [default : declaration ;]
        };
```

case-list:
 case *case-ident* : *declaration* ;
 case *case-ident* : *declaration* ; *case-list*

discriminant-declaration:
 declaration

The union definition looks like a cross between a C-union and a C-switch. An example:
```
        union net_object switch (net_kind kind) {
        case MACHINE:
                struct sockaddr_in sin;
        case USER:
                int uid;
        default:
                string whatisit;
        };
```

Compiles into:
```
        struct net_object {
                net_kind kind;
                union {
                        struct sockaddr_in sin;
                        int uid;
                        char *whatisit;
                } net_object;
        };
        typedef struct net_object net_object;
```

Note that the name of the union component of the output struct is the same as the name of the type itself.

Program Definitions

program-def:

```
        program program-ident {
                version-list
        } = program-number ;
```

version-list:

```
        version
        version version-list
```

version:

```
        version version-ident {
                procedure-list
        } = version-number ;
```

procedure-list:

```
        procedure-declaration
        procedure-declaration procedure-list
```

procedure-declaration:

```
        type-name procedure-ident ( type-name ) = procedure-number ;
```

The following is an example of a program definition to create a server that can get or set the date. The declaration looks as follows:

```
        program DATE_PROG {
                version DATE_VERS {
                        date DATE_GET(timezone) = 1;
                        void DATE_SET(date) = 2;   /* Greenwich mean time */
                } = 1;
        } = 100;
```

In the header file, this compiles into the following:

```
        #define DATE_PROG 100
        #define DATE_VERS 1
        #define DATE_GET 1
        #define DATE_SET 2
```

These **#define**'s are intended for use by the client program to reference the remote procedures.

If the server is being compiled using *rpcgen*, there are some important things to know. The server interfaces to the user's local procedures using a C function with the name as the program definition, but in lowercase and followed by the version number. Here is the local procedure that implements DATE_GET:

```
date *                  /* always returns a pointer to the results */
date_get_1(tz)
timezone *tz;           /* always takes a pointer to the arguments */
{
        static date d;  /* must be static! */

        /*
         * figure out the date
         * and store it in d
         */
        return(&d);
};
```

The name of the routine is the same as the **#define**'d name, but in all lowercase letters and followed by the version number. XDR recursively frees the argument after getting the results from the local procedure, so any data needed between calls should be copied from the argument. However, XDR neither allocates nor frees the results. The user must take care of the storage.

Make Inference Rules For Compiling XDR Headers

It is possible to set up suffix transformation rules in *make*(1) for compiling XDR routines and header files. The convention is that RPCL protocol files have the extension The *make* rules to do this are:

```
rpcgen -c $< -o $@

rpcgen -h $< -o $@
```

SEE ALSO

"RPC/XDR Tutorial" in the *CLIX System Guide*.

BUGS

Name clashes can occur when using program definitions, since the apparent scoping does not really apply. Most of these can be avoided by giving unique names for programs, versions, procedures and types.

NAME

rpipe - remote pipe program

SYNOPSIS

rpipe *host*.*user*[.[*password*]] *command-list*

DESCRIPTION

rpipe is a utility which transfers data to a command executing on a remote system via a pipe.

Host.*user*[.[*password*]] is the same syntax as that for *fmu*(1). The user is placed in the *user*'s home directory on *host*, and all commands executed have the same user and group ID as *user* on *host*. The standard input given to the *command-list* executed on *host* is the standard input from *rpipe*.

EXAMPLES

The following example copies the directory tree **src** from the local machine to the remote host **ipro3**, under the directory **/usr/test**.

```
find src -print | cpio -ovB | rpipe ipro3.sys "cd /usr/test; cpio -idumB"
```

WARNINGS

Characters with special meanings to the shell, like **(**, **)**, and **;** must be quoted.

BUGS

The standard output and standard error of *command-list* are not echoed back to the user.

NAME

rtape - remote tape manipulation program

SYNOPSIS

rtape *host tape-device command* [*count*]

DESCRIPTION

rtape performs a variety of functions on a remote tape. The only requirements of the remote system are that the tape is physically mounted on the tape drive and the tape drive is online.

Host is the name or address of the machine where the tape drive resides. *tape-device* is the name of the tape drive on the host machine. If *tape-device* is a CLIX non-rewindable device, the tape will not rewind after the command is completed.

Count is the size (in 512-byte blocks) of write operations and the number of files or records in skip operations. By default, the block size is set to 20, and the number of files or records to be skipped is one.

rtape uses standard input and standard output, so common tape commands such as *tar*(1) and *cpio*(1) can be used.

The following *commands* may be used:

read	Read the tape until an end-of-file mark is found and then rewind the tape unless a no-rewind device is specified. (*Count* is ignored).
write	Write the tape. If *count* is specified, *count* blocks are written. Two end-of-file marks are written to the tape when the write terminates and the tape rewinds unless a no-rewind device is specified.
read_nrw	Read the tape until an end-of-file mark is found, but do not rewind the tape. (*Count* is ignored).
write_nrw	Write the tape. If *count* is specified, *count* blocks are written. Two end-of-file marks are written to the tape when the write terminates, and the tape will be positioned between these two marks.
rew	Rewind the tape. (*Count* is ignored).
fsf	Forward skip *count* files on the tape.
fsr	Forward skip *count* records on the tape.
bsf	Backward skip *count* files on the tape.
bsr	Backward skip *count* records on the tape.
eof	Write one end-of-file mark on the tape.
erase	Erase the tape from the current position onward. (This feature is not supported on most tape controllers.)

examine Examine the tape, reporting the size of records and tape marks encountered. If *count* is zero, the tape is read until it is interrupted or the physical end-of-tape is encountered. If *count* is one, the tape is read until a single end-of-file mark is encountered and then rewound. If *count* is two or omitted, the tape is read until a double end-of-file (end-of-tape) is encountered.

The **read_nrw** and **write_nrw** commands are not supported on CLIX *host* machines. The **read** and **write** commands should be used instead with a no-rewind *tape-device* specified.

EXAMPLES

All examples assume a machine named "ipro3" with a tape drive attached to "/dev/rmt/mt5".

The following command could be used to make a *tar*(1) tape:

```
tar cvf - . | rtape ipro3 /dev/rmt/mt5 write
```

The following command could be used to read a *tar*(1) tape:

```
rtape ipro3 /dev/rmt/mt5 read | tar xvf -
```

The following command could be used to make a *cpio*(1) tape:

```
find . -print | cpio -ovB | rtape ipro3 /dev/rmt/mt5 write 10
```

The following command could be used to read a *cpio*(1) tape:

```
rtape ipro3 /dev/rmt/mt5 read | cpio -ivdumB
```

If "/dev/rmt/mt5n" is used, the tape will not rewind. This will allow for multiple archives on the same tape.

SEE ALSO

cpio(1), rtc(1), tar(1).
rtc(7S), tc(7S) in the *CLIX System Administrator's Reference Manual.*

WARNINGS

Some *commands* are not supported by all tape drives.

Some newer tape drives are not supported.

BUGS

Since this program uses a network, the remote system controls the tape. If the program is interrupted, the remote system attempts to recover. However, sometimes the recovery may be slow.

NAME

rtc - remote tape control

SYNOPSIS

rtc -a [**-s** *system*] [**-r** *rewdev*] [**-n** *norewdev*] [**-t** *timeout*] *controldev*
rtc -d *controldev*

DESCRIPTION

rtc allows a tape drive on another machine to be used as if it resides on the local machine. The following command options allow the tape drive to be configured:.

-a	Allocate a tape drive on a remote machine.
-d	Deallocate a tape drive on a remote machine.

The **-a** option allows the use of a tape drive on a remote machine. Once allocated, the remote tape drive remains allocated until a timeout occurs or the **-d** option is invoked. If the tape drive is being used when the **-d** option is invoked, an error is printed. The *controldev* parameter is the name of a tape control device (such as **/dev/rmt/rt0.ctl**) that controls the functions of other tape devices in the same group.

The following options are supported:

-s *system*	Specify the remote *system* where the tape drive resides.
-r *rewdev*	Specify the tape drive *rewdev* on *system* as the rewindable tape device.
-n *norewdev*	Specify the tape drive *norewdev* on *system* as the no-rewind tape device.
-t *timeout*	Set the idle time allowed to *timeout* minutes. If the tape is idle for *timeout* minutes, it is deallocated following a warning, which is printed on the system console. The default *timeout* is five minutes.

If an allocate option is not present, the corresponding environment variables are used if set:

RTCSYSTEM	Specify the remote system (**-s** option).
RTCREWIND	Specify the rewind device (**-r** option).
RTCNOREWIND	Specify the no-rewind device (**-n** option).
RTCTIMEOUT	Specify the idle timeout (**-t** option).

Since the remote tape driver that *rtc* uses, *rtc*(7S), is a STREAMS driver, the driver must be kept open for the network connection to be preserved. After issuing an allocate command, *rtc* forks a child process that remains in the background before returning to the user. If this process is killed or the user logs out, the remote tape drive will automatically be deallocated when the current process using the tape drive exits.

The **-d** option will cleanly close the connection to the remote host and send a signal to the process that was executed by the **-a** option.

rtc also looks in a series of **.rtcrc** files to determine the proper action if an option to the allocate command is not present. Given *controldev*, *rtc* looks for a line that begins with the name *controldev*. The line that begins with *controldev* should be followed by a remote system name, separated by spaces or tabs. *rtc* then continues looking for a line that begins with that remote system name. The line should contain a list of options in the following order:

```
rewdev [ norewdev [ timeout ] ]
```

The following is a sample file:

```
/dev/rmt/rt0.ctl iprol

iprol   /dev/rmt/mt6 /dev/rmt/mt6n        10
```

The files are checked as follows:

```
.rtcrc
$HOME/.rtcrc
/etc/.rtcrc
```

EXAMPLES

This command allocates the "/dev/rmt/mt6" tape drive on the machine "iprol".

```
rtc -a -s iprol -r /dev/rmt/mt6 -n /dev/rmt/mt6n -t 10 /dev/rmt/rt0.ctl
```

FILES

/dev/rmt/rt?	rewind *rtc* device
/dev/rmt/rt?n	no-rewind *rtc* device
/dev/rmt/rt?.ctl	control device (used only by *rtc*)

SEE ALSO

cpio(1), mt(1), tar(1), rtc_allocate(3N).
rtc_s(1M), rtc(7S), tc(7S) in the *CLIX System Administrator's Reference Manual.*

NAME

ruptime - show host status for each machine on the local network

SYNOPSIS

ruptime [-altur]

DESCRIPTION

ruptime gives a status line for each machine on the local network running *rwhod*(1M). The status is a summary of the current activity on each system. This data is provided by packets broadcast by each host once every three minutes.

Machines for which a status report has not been received in the last five minutes are shown as being down.

Users idle an hour or more are omitted from the listing unless the **-a** option is given.

Normally, the listing is sorted by host name. The **-l**, **-t**, and **-u** options specify sorting by load average, uptime, and number of users, respectively. The **-r** option reverses the sort order.

FILES

/usr/spool/rwho/whod.*	data files containing information about each machine

SEE ALSO

rwho(1).
rwhod(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

rwho - lists users logged in to machines on the local network

SYNOPSIS

rwho [-a]

DESCRIPTION

rwho produces output similar to *who*, but for all machines on the local network running *rwhod*(1M). If a report has not been received from a machine for five minutes, *rwho* assumes the machine is down and does not report users last known to be logged into that machine.

If a user has not typed to the system for a minute or more, *rwho* reports this idle time. If a user has not typed to the system for an hour or more, the user will be omitted from the output of *rwho* unless the **-a** option is given.

FILES

/usr/spool/rwho/whod.*	data files containing information about each machine

SEE ALSO

ruptime(1).
rwhod(1M) in the *CLIX System Administrator's Reference Manual.*

BUGS

This is unwieldy when the number of machines on the local network is large.

NAME

sccstorcs - build RCS file from SCCS file

SYNOPSIS

sccstorcs [-t] [-v] **s.***file* ...

DESCRIPTION

sccstorcs builds a Revision Control Systemk (RCS) file from each Source Code Control System (SCCS) file argument. The deltas and comments for each delta are preserved and installed in the new RCS file in order. Also preserved are the user access list and descriptive text from the SCCS file.

The following options are available:

-t Trace only. Prints detailed information about the SCCS file and lists the commands that would be executed to produce the RCS file. No commands are actually executed and no RCS file is created.

-v Verbose. Prints each command run while it is building the RCS file.

FILES

For each **s.***somefile*, *sccstorcs* writes the files *somefile* and *somefile*,**v**, which should not exist. *sccstorcs* will abort rather than overwrite files if they exist.

SEE ALSO

ci(1), co(1), rcs(1).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

DIAGNOSTICS

All diagnostics are written to **stderr**. Nonzero exit status on error.

BUGS

sccstorcs does not preserve all SCCS options specified in the SCCS file. Most notably, it does not preserve removed deltas, MR numbers, and cutoff points.

sccstorcs fails if used on an SCCS file with removed deltas.

sccstorcs fails if it checks in one version of an RCS file, gets the next SCCS version, and checks in the new version, all in the same second. This is due to *ci*(1) not allowing checkins with the same time stamp.

IDENTIFICATION

Ken Greer
Copyright © 1983 by Kenneth L. Greer

NAME

scpio - multibuffering and asynchronous I/O *cpio*(1)

SYNOPSIS

scpio -o [acBvV] [**-C** *bufsize*] [**-z** *bufcount*] [[**-O** *file*] [**-M** *message*]]

scpio -i [BcdmrtuvVfsSb6] [**-C** *bufsize*] [**-z** *bufcount*] [[**-I** *file*] [**-M** *message*]] [*pattern* ...]

scpio -p [adlmuvV] *directory*

scpio -x [cstvfB6] [**-C** *bufsize*] [**-z** *bufcount*] [**-I** *file*] [*pattern* ...]

DESCRIPTION

scpio is a modified version of *cpio*(1) with two major enhancements—asynchronous I/O and multibuffering—which significantly improve its performance. See *cpio*(1) for a description of basic capabilities and options.

In addition to the three standard modes of *cpio*(1) (input, output, and passthrough), *scpio* provides a verify mode (**-x** option). In this mode, *scpio* verifies the integrity of the archived file by comparing it to the input file. An error message is printed if a difference is found.

Additional options supported by *scpio* are as follows:

-x	Compare the archive and the input files and report any differences.
-z *bufcount*	Specify the number of buffers to be used for multibuffering. Permissible values are 2 to 25. If this parameter is not specified, it defaults to 1 and I/O is synchronous.

EXAMPLES

The following command uses 25 buffers, each 63488 bytes in size, to create an archive on **/dev/rmt/0m**:

```
scpio -o -C 63488 -z 25 -O /dev/rmt/0m
```

The next command reads an archive from **/dev/rmt/0m** using the same buffer count and size as the previous example:

```
scpio -idm -C 63488 -z 25 -I /dev/rmt/0m
```

SEE ALSO

cpio(1).

NOTES

When an archive is being verified (**-x** option), file headers and data are compared. If one of the input files is accessed between the time it was written and the time it is read for verification, the date in the header is changed. This results in a verification error, although nothing is wrong with the file.

Using the **-v** option may give less than optimal speed due to time spent writing file names to the screen. The **-V** option prints a dot for every file, verifying activity without the overhead of printing every file name.

NAME

sdb - symbolic debugger

SYNOPSIS

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

DESCRIPTION

sdb is a symbolic debugger that can be used with C and FORTRAN programs. It may be used to examine their object files and core files and provide a controlled environment for their execution.

Objfil is an executable program file that has been compiled with the **-g** (debug) option. If it has not been compiled with the **-g** option, the symbolic capabilities of *sdb* will be limited, but the file can still be examined and the program debugged. The default for *objfil* is **a.out**. *Corfil* is assumed to be a *core*(4) image file produced after executing *objfil*; the default for *corfil* is **core**. The *core*(4) file need not be present. A - in place of *corfil* will force *sdb* to ignore any *core*(4) image file. The colon-separated list of directories (*directory-list*) is used to locate the source files used to build *objfil*.

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

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

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

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

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

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

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

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

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

Addresses

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

$$b1 \leqslant address < e1 \rightarrow file\ address = address + f1 - b1$$

otherwise

$$b2 \leqslant address < e2 \rightarrow file\ address = address + f2 - b2$$

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

The initial setting of both mappings is suitable for normal *a.out*(4) and *core*(4) files. If either file is not the kind expected then, for that file; *b1* is set to 0; *e1* is set to the maximum file size; and *f1* is set to 0. In this way, the whole file can be examined with no address translation.

For *sdb* to be used on large files, all appropriate values are kept as signed, 32-bit integers.

Commands

The commands for examining data in the program are as follows:

t Print a stack trace of the terminated or halted program. This command will not work properly unless the program has been compiled to use a stack frame pointer (**-g** or **-ga** compiler options).

T Print the top line of the stack trace.

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

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

Legal values for *m* are:

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

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

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

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

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

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

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

X Print the current machine-language instruction.

The commands for examining source files are as follows:

e *procedure*
e *file-name*
e *directory/*
e *directory file-name*
The first two forms set the current file to the file containing *procedure* or to *file-name*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The

default is the current working directory. The latter two forms change the value of *directory*. If no procedure, file name, or directory is given, the current procedure name and file name are reported.

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

?regular expression?
: Search backward from the current line for a line containing a string matching *regular expression* as in *ed*(1). The trailing ? may be deleted.

p Print the current line.

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

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

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

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

count−
: Retreat the current line by *count* lines. Print the new current line.

The commands for controlling the source program execution are as follows:

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

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

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

s *count*
S *count*
Single step the program through *count* lines. If no count is given, the program runs for one line. **S** is equivalent to **s** except that it steps through procedure calls and **s** does not.

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

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

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

k Kill the program being debugged.

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

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

B Print a list of the currently-active breakpoints.

linenumber **d**
Delete a breakpoint at the given line. If no *linenumber* is given, the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from the standard input. If the line begins with a **y** or **d**, the breakpoint is deleted.

D Delete all breakpoints.

l Print the last executed line.

linenumber **a**
(Announce) If *linenumber* has the form *proc:number*, the command effectively executes a *linenumber* **b l**. If *linenumber* has the form *proc:*, the command effectively executes a *proc:* **b T**.

Miscellaneous commands:

!*command*
The command is interpreted by *sh*(1).

newline
If the previous command printed a source line, advance the current line by one line and print the new current line. If the previous command displayed a memory location, display the next memory location.

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

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

M Print the address maps.

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

" *string*
Print the given string. The C escape sequences with the form *character* are recognized, where *character* is a non-numeric character.

q Exit the debugger.

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

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

FILES

a.out
core

SEE ALSO

cc(1), f77(1), a.out(4), core(4).
syms(4) in the *UNIX System V Programmer's Reference Manual.*
sh(1) in the *UNIX System V User's Reference Manual.*

WARNINGS

When *sdb* prints the value of an external variable that has no debugging information, a warning is printed before the value. The size is assumed to be an integer.

Data stored in text sections cannot be distinguished from functions.

Line number information in optimized functions is unreliable, and some information may be missing.

BUGS

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

sdb cannot print the value of a FORTRAN parameter. It will erroneously print the address.

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

The range of a FORTRAN array subscript is assumed to be 1 to *n*, where *n* is the dimension corresponding to that subscript. This is only significant when the user omits a subscript, or uses * to indicate the full range. Arrays having subscripts whose lower bounds are not 1 produce no problems.

NAME

sethost - DNP remote login DECnet or CLIX node.

SYNOPSIS

sethost [-**hrx**] [-**e***string*]
sethost [-**hrx**] [-**e***string*] *node-name*
sethost [-**hrx**] [-**e***string*] [*node-area.*]*node-number*

DESCRIPTION

sethost allows a user to log in to a remote host supporting the Digital Network Architecture (DNA) from the local host. *sethost* uses the DNA heterogeneous Command Terminal Protocol (CTERM). *sethost* connects the user to any DECnet host that supports the CTERM protocol.

If *node-name* is specified, it must be one of the remote node names defined in the local *ncp*(1M) database. *Node-name* is a group of not more than six alphanumeric characters where the first character is a letter.

Alternately, an address of an active remote node may be specified. This address can specify an optional *area-number* range of 1-63. The *node-number* ranges from 1-1023 and must be unique in the network area. If the remote node is in the same network area as the local node, the *node-number* can be used alone.

If a node or address is not provided, *sethost* prompts for input.

Once the connection is established, the user is prompted from the remote node for login information.

The following options are available to *sethost*:

-h Display a brief help summary.

-r Display the release numbers of *sethost*(1).

-x Set the CTERM flow control passthrough characteristic (remoteflow) to TRUE. This allows <CONTROL>-S/<CONTROL>-Q to be passed to the remote instead of using local flow control. This is useful for a remote application, such as emacs, that uses <CONTROL>-S. If the user requires this option frequently, the environment variable, **REMOTEFLOW**, may be created with any non-NULL value. Users can refer to the environment manipulation routines for their particular shell. The default is to allow local flow control. When the user specifies the **-x** option, the remote performs all XON/XOFF processing. This may cause a delay when <CONTROL>-S/<CONTROL>-Q is used to stop output from an application that is writing to the screen, such as DIR or TYPE on VMS and *ls*(1) or *cat*(1) on CLIX.

-**e***string* Change the default escape string from ~. to *string*. There can be no spaces between **-e** and *string*. The *string* can be up to 10 characters long. Characters after 10 are ignored. Alternately, the user may set the escape string with the **ESCAPESTR** environment variable. If the **-e** option is not used and there is no environment variable, *sethost* uses the default ~..

The escape string allows the user to leave the *sethost* session at any time, even if the user is hung in a remote application program. Additionally, when connected to a remote VMS machine at the Digital Command Language (DCL) command line, the user can type two quick (within one second), consecutive <CONTROL>-Y or <CONTROL>-C characters to gain control. When the escape string or control characters are typed, the following question appears:

Do you wish to abort the network virtual terminal session?

If the user answers **Y**, the connection is terminated. Otherwise, the escape string or control characters are passed to the remote.

When connecting to VMS nodes, *sethost* allows command-line editing at the DCL prompt. The cursor movement assumes that a VT*x*00-style terminal is being used. If a non-VT*x*00 terminal is used, command-line editing may produce unexpected output. The left, right, up, and down arrow keys, along with the following control characters, are supported:

<CONTROL>-A	Toggle insert/overstrike mode.
<CONTROL>-E	Move cursor to end of line.
<CONTROL>-F	Move cursor right one space.
<CONTROL>-H	Move cursor to beginning of line.
<CONTROL>-R	Redisplay input.
<CONTROL>-U	Delete from cursor to beginning of line.
<CONTROL>-W	Delete word to left.
<CONTROL>-X	Functions same as <CONTROL>-U.

The default mode is overstrike. The mode is changed to insert on the command line by typing <CONTROL>-A. Changing the VMS overstrike/insert characteristic has no effect. After <RETURN> is pressed, the mode reverts to overstrike.

NOTES

VMS versions before V4.0 and DECnet-11M-PLUS versions before V3.0 do not support CTERM. Consequently, *sethost* is not supported on these machines.

SEE ALSO

sh(1), ksh(1), csh(1), getenv(3).
sethostd(1M), ncp(1M) in the *CLIX System Administrator's Reference Manual.*

CAVEATS

^I is displayed if the user presses the tab key when connected to a VMS machine.

NAME

showfiles - CRM utility for monitoring open files

SYNOPSIS

/usr/ip32/crm/showfiles [**-cdf**] [*input-option*] [**-o** *output-file*]

DESCRIPTION

showfiles displays a list of all processes on the system. It also lists open files for each process. Device names and i-node numbers are provided for each open file.

Once an open file's i-node number is determined, the *ncheck*(1M) command can be used to generate the path names from the i-node numbers. Refer to the System V Online manuals for more information on *ncheck*(1M).

The following options are are available:

-c	Run continuously.
-d	Translate major and minor device numbers to device names.
-f	Spawn *ncheck*(1M) to translate i-node numbers to file names. Also translate device numbers to to device names as with the **-d** option.
-o *output-file*	Direct output to *output-file*. A - for *output-file* directs output to **stdout**.

The following *input-options* are available:

-p *pid*	Specify the ID number of the process to monitor (PID). The user can key in **ps -e** at the system prompt to determine the PID of a process already running.
-n *process-name*	Specify the name of the process to monitor. The user can key in **ps -e** at the system prompt to determine the name of a process already running.
-i *input-file*	Read the data from *input-file* each interval. The *input-file* must have been created as an *output-file* using the **-o** option. A - for *input-file* reads input from standard input.

SEE ALSO

crm(1).
ncheck(1M) in the *UNIX System V Administrator's Reference Manual.*

WARNINGS

Sending raw data to a file can create a very large file.

Device name translation takes more time.

File name lookup by *ncheck*(1M) takes much more time, since each file system must be searched for the i-node numbers.

NAME

showmemory - CRM utility for monitoring process memory regions

SYNOPSIS

/usr/ip32/crm/showmemory [*input-option*]

DESCRIPTION

showmemory displays a list of all processes on the system. It also lists attached memory regions associated with each process.

The following *input-options* are available:

-p *pid*	Specify the ID number of the process to monitor (PID). The user can key in **ps -e** at the system prompt to determine the PID of a process already running.
-n *process-name*	Specify the name of the process to monitor. The user can key in **ps -e** at the system prompt to determine the name of a process already running.
-e *command arg ...*	Run, provide arguments for, and monitor a program.

A brief explanation of the *showmemory* information given for each displayed region follows:

REGION TYPE — Displays the region type, which can be one of the following types:

TEXT	main executable code
DATA	main data region
STACK	process stack
SHMEM	shared memory
DMM	double mapped memory
LIBTXT	shared library code
LIBDAT	shared library data

REGION NUMBER	Displays the CLIX internal identification number of the region. If a region number displays in more than one process, the region is shared among those processes.
VIRTUAL SIZE	Displays the amount of virtual memory allocated to the region. The virtual size of the regions is also allocated from the available swap space.
PHYSICAL SIZE	Displays the amount of real memory currently being used by the region.
SHARED	Displays the number of processes currently attached to the region. If no number appears in the shared column, the region is being used only by the one process.
PERCENT MEMORY	Displays the percentage of physical memory allocated to the region. This number is weighted by the number

	of processes sharing the region.
CUMULATIVE	Displays the cumulative percentage of physical memory used by the regions.

After the list of processes has been displayed, a system summary is displayed. The summary gives the following information:

Total Physical Memory On System
: Displays (in megabytes) the amount of real memory the system has.

Used By Processes
: Displays the final cumulative total of physical memory being used by the process regions.

Process Overhead
: Displays the amount of memory used by the system to keep page tables and user blocks.

Unattached Regions
: Displays the amount of physical memory being used by regions with unattached processes. Unattached regions can occur when a program has the sticky bit set in its mode (see *chmod*(1)).

Available Memory
: Displays the amount of physical memory that is available for processes to use.

Initial Clix Size
: Displays the amount of physical memory used by the CLIX operating system when the system boots.

Allocated By Clix
: Displays the amount of physical memory allocated when the CLIX operating system is running. For example, when a driver is loaded, its text and data section occupies a section of physical memory.

SEE ALSO

crm(1), chmod(1).

NAME

stty - set the options for a terminal

SYNOPSIS

stty [**-a**] [**-g**] [*option* ...]

DESCRIPTION

stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

In this report, if a character is preceded by a caret (^), the value of that option is the corresponding <CONTROL> character (e.g., "^h" is <CONTROL>-H; in this case, <CONTROL>-H is the same as the <BACK SPACE> key). The sequence ^' means that an option has a null value. For example, normally **stty -a** will report that the value of **swtch** is ^'; however, if *shl*(1) or *layers*(1) has been invoked, **stty -a** will have the value "^z".

-a	Reports all option settings.
-g	Reports current settings in a form that can be used as an argument to another *stty* command.

Options in the last group are implemented using options in the previous groups. Note that many combinations of options do not make sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (-parenb)	Enable (disable) parity generation and detection.
parodd (-parodd)	Select odd (even) parity.
cs5 cs6 cs7 cs8	Select character size (see *termio*(7S)).
0	Hang up phone line immediately.
110 300 600 1200 1800 2400 4800 9600 19200 38400	Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)
hupcl (-hupcl)	Hang up (do not hang up) dataphone connection on last close.
hup (-hup)	Same as **hupcl (-hupcl)**.
cstopb (-cstopb)	Use two (one) stop bits per character.
cread (-cread)	Enable (disable) the receiver.
clocal (-clocal)	Assume a line without (with) modem control.
loblk (-loblk)	Block (do not block) output from a noncurrent layer.

Input Modes

ignbrk (-ignbrk)	Ignore (do not ignore) break on input.
brkint (-brkint)	Signal (do not signal) INTR on break.
ignpar (-ignpar)	Ignore (do not ignore) parity errors.
parmrk (-parmrk)	Mark (do not mark) parity errors (see *termio*(7S)).
inpck (-inpck)	Enable (disable) input parity checking.
istrip (-istrip)	Strip (do not strip) input characters to seven bits.
inlcr (-inlcr)	Map (do not map) NL to CR on input.
igncr (-igncr)	Ignore (do not ignore) CR on input.
icrnl (-icrnl)	Map (do not map) CR to NL on input.
iuclc (-iuclc)	Map (do not map) uppercase alphabetics to lowercase on input.
ixon (-ixon)	Enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.
ixany (-ixany)	Allow any character (only DC1) to restart output.
ixoff (-ixoff)	Request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (-opost)	Post-process output (do not post-process output; ignore all other output modes).
olcuc (-olcuc)	Map (do not map) lowercase alphabetics to uppercase on output.
onlcr (-onlcr)	Map (do not map) NL to CR-NL on output.
ocrnl (-ocrnl)	Map (do not map) CR to NL on output.
onocr (-onocr)	Do not (do) output CRs at column zero.
onlret (-onlret)	On the terminal, NL performs (does not perform) the CR function.
ofill (-ofill)	Use fill characters (use timing) for delays.
ofdel (-ofdel)	Fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	Select style of delay for carriage returns (see *termio*(7S)).
nl0 nl1	Select style of delay for linefeeds (see *termio*(7S)).
tab0 tab1 tab2 tab3	Select style of delay for horizontal tabs (see *termio*(7S)).
bs0 bs1	Select style of delay for backspaces (see *termio*(7S)).

ff0 ff1	Select style of delay for formfeeds (see *termio*(7S)).
vt0 vt1	Select style of delay for vertical tabs (see *termio*(7S)).

Local Modes

isig (-isig)	Enable (disable) the checking of characters against the special control characters INTR, QUIT, SWTCH, and SUSP.
icanon (-icanon)	Enable (disable) canonical input (ERASE and KILL processing).
xcase (-xcase)	Canonical (unprocessed) uppercase/lowercase presentation.
echo (-echo)	Echo back (do not echo back) every character typed.
echoe (-echoe)	Echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEd character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (-echok)	Echo (do not echo) NL after KILL character.
lfkc (-lfkc)	The same as **echok (-echok)**; obsolete.
echonl (-echonl)	Echo (do not echo) NL.
noflsh (-noflsh)	Disable (enable) flush after INTR, QUIT, SWTCH, or SUSP.
tostop (-tostop)	Stop (do not stop) background jobs if they attempt terminal output.
stwrap (-stwrap)	Disable (enable) truncation of lines longer than 79 characters on a synchronous line.
stflush (-stflush)	Enable (disable) flush on a synchronous line after every *write*(2).
stappl (-stappl)	Use application mode (use line mode) on a synchronous line.

Control Assignments

control-character c	Set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **swtch**, **susp**, **eof**, **eol**, **ctab**, **min**, or **time**. (**ctab** is used with **-stappl**; **min** and **time** are used with **-icanon**; see *termio*(7S).) If *c* is preceded by a caret (^) (escaped from the shell), the value used is the corresponding <CONTROL> character (e.g., "^d" is a <CONTROL>-D); "^?" is interpreted as <DEL> and "^-" is interpreted as undefined.

line *i*	Set line discipline to *i* ($0 < i < 127$).

Combination Modes

evenp or **parity**	Enable **parenb** and **cs7**.
oddp	Enable **parenb**, **cs7**, and **parodd**.
-parity, **-evenp**, or **-oddp**	Disable **parenb**, and set **cs8**.
raw (**-raw** or **cooked**)	Enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, SUSP, EOT, or output post processing).
nl (**-nl**)	Unset (set) **icrnl**, **onlcr**. In addition **-nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (**-lcase**)	Set (unset) **xcase**, **iuclc**, and **olcuc**.
LCASE (**-LCASE**)	Same as **lcase** (**-lcase**).
tabs (**-tabs** or **tab3**)	Preserve (expand to spaces) tabs when printing.
ek	Reset ERASE and KILL characters back to normal **#** and **@**.
sane	Reset all modes to reasonable values.
term	Set all modes suitable for the terminal type *term*, where *term* is **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

SEE ALSO

termio(7S) in the *CLIX System Administrator's Reference Manual.*
ioctl(2) in the *UNIX System V Programmer's Reference Manual.*

NAME

telnet - user interface to the TELNET protocol

SYNOPSIS

telnet [*host* [*port*]]

DESCRIPTION

telnet communicates with another host using the TELNET protocol. If *telnet* is invoked without arguments, it enters command mode, indicated by its prompt (**telnet>**). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command (see below) with those arguments.

Once a connection is opened, *telnet* enters an input mode. The input mode entered will be either *character-at-a-time* or *line-by-line*, depending on what the remote system supports.

In *character-at-a-time* mode, most text typed is immediately sent to the remote host for processing.

In *line-by-line* mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The local echo character (initially <CONTROL>-E) will turn the local echo off and on. (This would mostly be used to enter passwords without the password being echoed.)

In either mode, if the **localchars** toggle is TRUE (the default in *line-by-line* mode as shown below), the user's QUIT and INTR characters are trapped locally and sent as TELNET protocol sequences to the remote side. Some options (**toggle autosynch** below) cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of **quit** and **intr**).

While connected to a remote host, the user may enter **telnet** command mode may by keying in the **telnet** escape character (initially <CONTROL>-]). In command mode, the normal terminal editing conventions are available.

Commands

The following commands are available. Only enough of each command to uniquely identify it needs to be typed. (This is also true for arguments to the **mode**, **set**, **toggle**, and **display** commands).

open *host* [*port*]
: Open a connection to the named host. If a port number is not specified, *telnet* will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see *hosts*(4)) or an Internet address specified in the dot notation (see *inet*(3B)).

close Close a TELNET session and return to command mode.

quit Close any open TELNET session and exit *telnet*. An end-of-file (in command mode) will also close a session and exit.

z Suspend *telnet*. This command works only when using *csh*(1) or *ksh*(1).

mode *type*
Type is either **line** (for *line-by-line* mode) or **character** (for *character-at-a-time* mode). The remote host is asked for permission to enter the requested mode. If the remote host can enter that mode, the requested mode will be entered.

status Show the current status of *telnet*. This includes the peer the user is connected to and the current mode.

display [*argument*...]
Display all or some of the **set** and **toggle** values (see below).

? [*command*]
Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information for that command only.

send *arguments*
Send one or more special character sequences to the remote host. The following arguments may be specified. (More than one argument may be specified at a time.)

escape Send the current *telnet* escape character (initially <CONTROL>-]).

synch Send the TELNET SYNCH sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as Transmission Control Protocol (TCP) urgent data. (This may not work if the remote system is a 4.2 Berkeley Software Distribution (BSD) system. If it does not work, a lowercase "r" may be echoed on the terminal).

brk Send the TELNET BRK (BReaK) sequence, which may be significant to the remote system.

ip Send the TELNET IP (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

ao Send the TELNET AO (Abort Output) sequence, which should cause the remote system to flush all output **from** the remote system **to** the user's terminal.

ayt Send the TELNET AYT (Are You There) sequence, to which the remote system may or may not choose to respond.

ec Send the TELNET EC (Erase Character) sequence, which should cause the remote system to erase the last character entered.

el Send the TELNET EL (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

ga Send the TELNET GA (Go Ahead) sequence, which likely is not significant to the remote system.

nop Send the TELNET NOP (No OPeration) sequence.

? Print help information for the **send** command.

set *argument value*
Set any one of a number of *telnet* variables to a specific value. The special value **off** turns off the function associated with the variable. The values of variables may be interrogated with the **display** command. The variables that may be specified are as follows:

echo This is the value (initially <CONTROL>-E) that, when in *line-by-line* mode, toggles between echoing entered characters locally (for normal processing), and suppressing echoing of entered characters (such as for entering a password).

escape This is the *telnet* escape character (initially <CONTROL>-E) that causes *telnet* to enter command mode (when connected to a remote system).

interrupt If *telnet* is in **localchars** mode (see **toggle localchars** below) and the INTR character is keyed in, a TELNET IP sequence (see **send ip** above) is sent to the remote host. The initial value for the interrupt character is interpreted as the terminal's INTR character.

quit If *telnet* is in **localchars** mode (see **toggle localchars** below) and the QUIT character is typed, a TELNET BRK sequence (see **send brk** above) is sent to the remote host. The initial value for the quit character is interpreted as the terminal's QUIT character.

erase If *telnet* is in **localchars** mode (see **toggle localchars** below) and *telnet* is operating in *character-at-a-time* mode, when this character is typed, a TELNET EC sequence (see **send ec** above) is sent to the remote system. The initial value for the erase character is interpreted as the terminal's ERASE character.

kill If *telnet* is in **localchars** mode (see **toggle localchars** below) and *telnet* is operating in *character-at-a-time* mode, when this character is typed, a TELNET EL sequence (see **send el** above) is sent to the remote system. The initial value for the kill character is interpreted as the terminal's KILL character.

eof If *telnet* is operating in *line-by-line* mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the end-of-file character is interpreted as the terminal's EOF character.

toggle *argument* ...
Toggle (between TRUE and FALSE) various flags that control how *telnet* responds to events. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are as follows:

localchars If this is TRUE, the INTR, QUIT, ERASE, and KILL characters (see **set** above) are recognized locally and transformed into appropriate TELNET control sequences (respectively, **ao**, **ip**, **brk**, **ec**, and **el**; see **send** above). The initial value for this toggle is TRUE in *line-by-line* mode and FALSE in *character-at-a-time* mode.

autosynch If **autosynch** and **localchars** are both TRUE, when the INTR or QUIT characters are typed the resulting TELNET sequence sent is followed by the TELNET SYNCH sequence. (See **set** above for descriptions of the INTR and QUIT characters.) This procedure should cause the remote system to begin discarding all previously typed input until both of the TELNET sequences have been read and acted on. The initial value of this toggle is FALSE.

crmod Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped to a carriage return followed by a line feed. This mode does not affect characters typed by the user; only those received from the remote host are affected. This mode is not useful unless the remote host only sends a carriage return, but it never sends a line feed. The initial value for this toggle is FALSE.

debug Toggle socket-level debugging. (This is useful only to the super-user.) The initial value for this toggle is FALSE.

options Toggle the display of some internal *telnet* protocol processing (concerning TELNET options). The initial value for this toggle is FALSE.

netdata Toggle the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

? Display the legal **toggle** commands.

SEE ALSO

visit(1), rlogin(1), inet(3B), hosts(4) in the *CLIX Programmer's and User's Reference Manual.*
telnetd(1M) in the *CLIX System Administrator's Reference Manual.*

CAVEATS

On some remote systems, echo must be turned off manually in *line-by-line* mode.

In *line-by-line* mode, the terminal's EOF character is recognized (and sent to the remote system) only when it is the first character on a line.

NAME

test - condition evaluation command

SYNOPSIS

test *expr*
[*expr* **]**

DESCRIPTION

test evaluates the expression *expr* and, if its value is true, sets a zero (true) exit status; otherwise, a nonzero (false) exit status is set. *test* also sets a nonzero exit status if there are no arguments. When permissions are tested, the effective process user ID is used.

All operators, flags, and brackets (brackets used as shown in the second **SYNOPSIS** line) must be separate arguments to the *test* command; normally these items are separated by spaces.

The following primitives are used to construct *expr*:

-r *file*	True if *file* exists and is readable.
-w *file*	True if *file* exists and is writable.
-x *file*	True if *file* exists and is executable.
-f *file*	True if *file* exists and is a regular file.
-d *file*	True if *file* exists and is a directory.
-c *file*	True if *file* exists and is a character special file.
-b *file*	True if *file* exists and is a block special file.
-L *file*	True if *file* exists and is a symbolic link.
-p *file*	True if *file* exists and is a named pipe (fifo).
-u *file*	True if *file* exists and its set-user-ID bit is set.
-g *file*	True if *file* exists and its set-group-ID bit is set.
-k *file*	True if *file* exists and its sticky bit is set.
-s *file*	True if *file* exists and has a size greater than zero.
-t [*fildes*]	True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
-z *s1*	True if the length of string *s1* is zero.
-n *s1*	True if the length of the string *s1* is nonzero.
s1 **=** *s2*	True if strings *s1* and *s2* are identical.
s1 **!=** *s2*	True if strings *s1* and *s2* are *not* identical.
s1	True if *s1* is *not* the null string.
n1 **-eq** *n2*	True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, and **-le** may be used instead of **-eq**.

These primaries may be combined with the following operators:

!	Unary negation operator.
-a	Binary AND operator.
-o	Binary OR operator (**-a** has higher precedence than **-o**).
(*expr*)	Parentheses for grouping. Notice also that parentheses are meaningful to the shell. Therefore, they must be quoted.

SEE ALSO

find(1).
sh(1) in the *UNIX System V User's Reference Manual.*

WARNINGS

If an owned file is tested (the **-r**, **-w**, or **-x** tests), but the permission tested does not have the owner bit set, a nonzero (false) exit status will be returned even though the file may have the group or other bit set for that permission. The correct exit status will be set if executed by the super-user.

The = and **!**= operators have a higher precedence than the **-r** through **-n** operators, and = and **!**= always expect arguments; therefore, = and **!**= cannot be used with the **-r** through **-n** operators.

If more than one argument follows the **-r** through **-n** operators, only the first argument is examined; the others are ignored unless **-a** or **-o** is the second argument.

NAME

tftp - trivial file transfer program

SYNOPSIS

/usr/ip32/tcpip/tftp [*host* [*port*]]

DESCRIPTION

tftp is the user interface to the Internet Trivial File Transfer Protocol (TFTP), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line. In this case *tftp* uses *host* as the default host for future transfers (see the **connect** command below). If *host* is specified on the command line, an optional *port* number can also be specified. In this case, *tftp* uses that *port* number for future transfers.

Once *tftp* is running, it issues the **tftp>** prompt and recognizes the following commands:

connect *host* [*port*]
: Set the *host* (and optionally *port*) for transfers. The TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers; thus, the **connect** command does not actually create a connection, but specifies the host to be used for transfers. The **connect** command does not need to be used; the remote host can be specified as part of the **get** or **put** command.

mode *transfer-mode*
: Set the mode for transfers; *transfer-mode* may be **ascii** or **binary**. The default is **ascii**.

put *file*
put *local-file remote-file*
put *file1 file2 ... fileN remote-directory*
: Copy a file or set of files to the specified remote file or directory. The destination can be in one of two forms: a *file-name* on the remote host if the host has been specified or a string of the form *host@file-name* to specify both a host and file name at once. If the latter form is used, the host name specified becomes the default for future transfers. If the *remote-directory* form is used, the remote host is assumed to be a UNIX machine.

get *file-name*
get *remote-name local-name*
get *file1 file2 ... fileN*
: Get a file or set of files from the specified source. The source can be in one of two forms: a *file-name* on the remote host if the host has been specified or a string of the form *host@file-name* to specify both a host and file name at once. If the latter form is used, the last host name specified becomes the default for future transfers.

quit Exit *tftp*. An end-of-file also exits.

verbose
Toggle verbose mode.

trace Toggle packet tracing.

status Show the current status.

rexmt *retransmission-timeout*
Set the per-packet retransmission timeout. The timeout is specified in seconds.

timeout *total-transmission-timeout*
Set the total transmission timeout. The timeout is specified in seconds.

ascii Synonym for **mode ascii**.

binary
Synonym for **mode binary**.

? [*command-name* ...]
Print help information.

help [*command-name* ...]
Print help information.

SEE ALSO

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

CAVEATS

Because no user login or validation is in the TFTP protocol, the remote site will probably have file access restrictions. The exact methods are specific to each site.

NAME

to_flop, fr_flop - continuous floppy disk filters

SYNOPSIS

to_flop [**-l**] [**-b** *blocks*] [**-n** *num*] [**-f** *name*] [**-d** *device*]

fr_flop [**-l**] [**-b** *blocks*] [**-n** *num*] [**-f** *name*] [**-d** *device*]

DESCRIPTION

to_flop copies data from standard input to a floppy disk device, prompting the user to insert sequential floppy disks as needed. Data is output in 512-byte blocks until the specified number of blocks is written to the floppy disk.

fr_flop reads data from a floppy disk device and outputs to standard out, prompting the user to insert sequential floppy disks as needed.

The following options are recognized:

-l	Indicate that the floppy has low density and contains only 720 blocks (1440 on a 3½-inch disk) (The default is high density containing 2400 (2880) blocks.)
-b *blocks*	Specify the total number of 512-byte blocks on the floppy. This option is used when the floppy does not contain the standard 720 (1440) or 2400 (2880) blocks.
-n *num*	Specify the starting number for subsequent floppy prompting. This is used only when generating the prompt message.
-f *name*	Specify the floppy set name for subsequent floppy prompting. This is used only when the prompt message is generated.
-d *device*	Specify that *device* will be used as the input or output device. If this option is not specified, **/dev/rdsk/fl** is used.

The **-d** and **-b** options allow the utility to be used with devices other than floppy disks if the device capacity is known.

EXAMPLES

To make a multiple floppy disk *cpio*(1) archive of the **/usr** file system, use the following command:

```
find /usr -print | cpio -o | to_flop
```

To retrieve the *cpio*(1) archive located on the floppy disk set made from the above *to_flop* example, use the following command:

```
fr_flop | cpio -ivmud
```

FILES

/dev/rdsk/f1	default floppy device

WARNINGS

Label the floppy disks created by *to_flop* with sequence numbers. No indication of the sequence number is written to the floppy disks.

NAME

topcpu - CRM utility for monitoring CPU time

SYNOPSIS

/usr/ip32/crm/topcpu [**-I** *interval*] [**-i** *input-file*] [**-o** *output-file*] [**-w**]

DESCRIPTION

topcpu monitors the amount of CPU time being used in each of the following modes: user, kernel, wait I/O, swap I/O, phys I/O (physical I/O), and sxbrk (time spent allocating memory for a new job).

The following options are available:

-I *interval*	Specify how frequently the monitor samples and displays information. The *interval* is the number of seconds. The default is 2.
-i *input-file*	Read the data from *input-file* each interval. The *input-file* must have been created as an *output-file* using the **-o** option. A - for *input-file* reads input from **stdin**.
-o *output-file*	Direct output to *output-file*. A - for *output-file* directs output to **stdout**.
-w	Execute *topcpu* in graphics-based format.

In graphics, to expand the list of processes being monitored, stretch the length of the window with the standard modify icon. To receive a description of each category represented in the monitor bar graphs, select the question mark (?) icon from the window icon box. A help window will appear.

In graphics, to change the colors of the bar graphs in *topcpu*, select the color palette icon from the window icon box. A small Color menu will appear. The foreground color is displayed when the menu first appears. Clicking the mouse button moves to the next color. Exit and save the changes by selecting the delete icon in the Colors window. These colors are saved for this monitoring session only.

A brief explanation of the *topcpu* fields follows. A similar list can be accessed online by keying in **?** while the monitor is running.

%Used	Displays the amount of overall CPU time being used by a process.
%User/System used	Displays a type of bar graph composed of **U** (user) and **S** (system) to illustrate visually how much CPU time used by a process is being taken up by the user (the process itself) or by the system. Each **U** or **S** represents about two percent.

SEE ALSO

crm(1).

WARNINGS

Sending raw data to a file can create a very large file.

NAME

topfault - CRM utility for monitoring page faults

SYNOPSIS

/usr/ip32/crm/topfault [-I *interval*] [-i *input-file*] [-o *output-file*]

DESCRIPTION

topfault monitors the page faults being encountered by each process running on the system.

The following options are available:

-I *interval*	Specify how frequently the monitor samples and displays information. The *interval* is the number of seconds. The default is 2.
-i *input-file*	Read the data from *input-file* each interval. The *input-file* must have been created as an *output-file* using the -o option. A - for *input-file* reads input from **stdin**.
-o *output-file*	Direct output to *output-file*. A - for *output-file* directs output to **stdout**.

A brief explanation of the *topfault* fields follows. A similar list can be accessed online by keying in a ? while the monitor is running.

Sample time — Displays how frequently (in seconds) the monitor gathers and displays information. The default setting is two seconds. This time interval can be changed by pressing the up arrow key (to increment) and the down arrow key (to decrement).

Max displayed — Displays the maximum number of faulting processes. This value can be changed by pressing the right arrow key (to increment) and the left arrow key (to decrement).

vfault — Displays virtual faults. The vfault value is the sum of the four following values defined by CLIX. Remember that, out of the four following types of faults, only swap and file faults go to the disk; demand and cache faults are satisfied in memory.

demand	demand zero and demand fill pages
swap	fault satisfied when swapping to memory
cache	fault satisfied in the cache
file	fault satisfied from a file

pfault — Displays protection faults. The pfault value is the sum of the following values:

cop_wrt	(Copy-on-write) If two processes are sharing a copy-on-write page in memory, the page must be copied when one process needs to

		write to the page.
	steal	If a page is marked copy-on-write but only one process is accessing it, the page does not need to be copied. Instead, the protections are changed on the page so that one process can write to it.

freedpgs	**Displays the number of pages that were freed on the system during the last sample interval.**
unmodsw	Displays the number of unmodified pages in swap (as determined by *getpages*) during the sample time period.
unmodfl	Displays the number of unmodified pages in all files (as determined by *getpages*) during the sample time period.
swapin	Displays the number of pages swapped into memory during the sample time period.
swapout	Displays the number of pages swapped out of memory during the sample time period.

SEE ALSO
crm(1).

WARNINGS
Sending raw data to a file can create a very large file.

NAME

topio - CRM utility for monitoring I/O activity

SYNOPSIS

/usr/ip32/crm/topio [**-I** *interval*] [**-i** *input-file*] [**-o** *output-file*]

DESCRIPTION

topio monitors the I/O activity on the system and displays which processes are performing the activity.

The following options are available:

-I *interval*	Specify how frequently the monitor samples and displays information. *Interval* is the number of seconds. The default is 2.
-i *input-file*	Read the data from *input-file* each interval. *Input-file* must have been created as an *output-file* using the **-o** option. A - for *input-file* reads input from **stdin**.
-o *output-file*	Direct output to *output-file*. A - for *output-file* directs output to **stdout**.

A brief explanation of the *topio* fields follows. A similar list can be accessed online by keying in **?** while the monitor is running.

b_read b_wrt	Displays the number of reads (b_read) and writes (b_wrt) to the block-oriented device (the disk). The "b" represents block.
1_read 1_wrt	Displays the number of data accesses (by a program) to the system buffer cache.
cache	Displays the percent of I/O that is satisfied by the buffer cache (rather than by the block-oriented device, or disk). This value is derived from the difference between the b_read and 1_read values.
phread phwrt	Displays the number of physical reads and writes to the raw disk.
sysrd syswrt	Displays the number of system calls to the read and write routines.
rdch wrtch	Displays the total number of bytes (characters) that are transferred by all read and write calls from a program regardless of where the data came from (cache, disk, or memory).
device	Displays the Small Computer System Interface (SCSI) devices involved in I/O on the system.

ops	Displays the number of I/O operations that occurred on the corresponding SCSI bus.
busy	Displays the percentage of time that the SCSI device was busy with I/O operations (versus how much time spent idle).
bcnt	Displays a count of disk blocks that were transferred.
avque	Displays the average number of times that I/O had to wait because the SCSI device was busy servicing other I/O requests.
currque	Displays the current I/O queue depth (how many I/O requests are in the queue to be serviced).
ioch	Displays the number of characters transferred by the corresponding process.

SEE ALSO

crm(1).

WARNINGS

Sending raw data to a file can create a very large file.

NAME

topmem - CRM utility for monitoring physical and virtual memory

SYNOPSIS

/usr/ip32/crm/topmem [**-I** *interval*] [**-i** *input-file*] [**-o** *output-file*] [**-w**]

DESCRIPTION

topmem monitors the amounts of physical and virtual memory being used by processes on the system.

The following options are available:

-I *interval*	Specify how frequently the monitor samples and displays information. *Interval* is the number of seconds. The default is 2.
-i *input-file*	Read the data from *input-file* each interval. *Input-file* must have been created as an *output-file* using the **-o** option. A - for *input-file* reads input from **stdin**.
-o *output-file*	Direct output to *output-file*. A - for *output-file* directs output to **stdout**.
-w	Execute *topmem* in graphics-based format.

In graphics, to expand the list of processes being monitored, stretch the length of the window with the standard modify icon. To receive a description of each category represented in the monitor bar graphs, select the question mark (?) icon from the window icon box. A help window will appear.

In graphics, to change the colors of the bar graphs in *topmem*, select the color palette icon from the window icon box. A small Color menu will appear. The foreground color is displayed when the menu first appears. Clicking the mouse button moves to the next color. Exit and save the changes by selecting the delete icon in the Colors window. These colors are saved for this monitoring session only.

A brief explanation of the *topmem* fields follows. A similar list can be accessed online by keying in **?** while the monitor is running.

freepages	Displays the average number of pages that were free (available) during the last sample interval.
proc_phys	Displays the sum of all Weighted_physical_size values. The resulting sum indicates the total physical memory used by all processes.
freeswap	Displays the amount of space available on the swap device.
Physical_size	Displays the total amount of physical memory (valid pages) being used by the indicated process.

Virtual_size	Displays the size of the virtual address space being used by the indicated process. This value indicates the amount of swap space being allocated to processes.
Weighted_physical_size	Displays the sum of valid pages used by a process, modified by the number of processes that share it. When several processes can share memory pages, fewer pages will need to be allocated for the later processes since they will share some of the pages that have already been allocated by earlier processes. This value indicates the amount of physical memory actually being used.
	For example, if three *vterm* processes were running, the first process executed would be allocated the memory pages needed to run. The second and third *vterm* processes would not require as many memory pages because they could share some of the pages allocated to the original process. Therefore, the weighted physical size of each process will vary depending on the number of pages already allocated to another process that the processes can share.

SEE ALSO

crm(1).

WARNINGS

Sending raw data to a file can create a very large file.

NAME

topsys - CRM utility for monitoring system activity

SYNOPSIS

/usr/ip32/crm/topsys [**-I** *interval*] [**-i** *input-file*] [**-o** *output-file*]

DESCRIPTION

topsys monitors the activity of the entire system. It simultaneously displays the activities that the other four system monitors (*topmem*(1), *topcpu*(1), *topio*(1), and *topfault*(1)) show individually to give a complete overview of system activity.

topsys is a graphics-based monitor in which the percentage of system resources being used by each process is represented by bars of contrasting colors. *topsys* cannot presently display in a curses-based format. Although the alphanumeric console of servers will not display graphics, *topsys* can run on a local server and display on a remote workstation.

The following options are available:

-I *interval*	Specify how frequently the monitor samples and displays information. *Interval* is the number of seconds. The default is 2.
-i *input-file*	Read the data from *input-file* each interval. *Input-file* must have been created as an *output-file* using the -o option. A - for *input-file* reads input from **stdin**.
-o *output-file*	Direct output to *output-file*. A - for *output-file* directs output to **stdout**.

In graphics, to expand the list of processes being monitored, stretch the length of the window with the standard modify icon. To receive a description of each category represented in the monitor bar graphs, select the question mark (?) icon from the window icon box. A help window will appear.

In graphics, to change the colors of the bar graphs in *topsys*, select the color palette icon from the window icon box. A small Color menu will appear. The foreground color is displayed when the menu first appears. Clicking the mouse button moves to the next color. Exit and save the changes by selecting the delete icon in the Colors window. These colors are saved for this monitoring session only.

EXAMPLES

To run *topsys* on a graphics monitor from a remote server, the following command may be used:

```
topsys -o - | rpipe node.user.password topsys -i -
```

In the above syntax, *node.user.password* is the node name and login of the graphics workstation on which to display the monitor.

SEE ALSO

crm(1), topmem(1), topcpu(1), topio(1), topfault(1).

WARNINGS

Sending raw data to a file can create a very large file.

NAME
ucpnice - run a process at UCP priority

SYNOPSIS
ucpnice *priority command* [*argument* ...]
ucpnice *priority* **-p** *pid*

DESCRIPTION
ucpnice executes *command* at the User Controlled Priority (UCP) specified by *priority*. If *priority* is 128, then the UCP priority is cleared.

If **-p** is specified, then the currently running process with process ID *pid* is changed to the UCP priority *priority*.

Additional processes that may be *fork*(2)ed from *command* or the process *pid* inherit the UCP priority of their parent.

This command fails if the *priority* is not in the range 0-128 or the process ID specified does not exist.

SEE ALSO
ucpset(2I), ucpclr(2I).

DIAGNOSTICS
ucpnice returns the exit status of the subject command.

NOTES
Only the super-user can execute this command.

NAME

visit - Intergraph remote login program

SYNOPSIS

visit [-**p** *protocol*] [*option* ...] [*host*]

DESCRIPTION

visit is a remote login program that supports the Intergraph Xerox Network Systems (XNS) XT protocol and the Bridge XNS Virtual Terminal Protocol (VTP). Both protocols can be active at the same time, providing the same user interface.

If *visit* is invoked without command-line options, it will enter interactive mode with a **visit>** prompt.

Once a connection is made, an escape sequence will return *visit* to interactive mode.

Host specifies the remote system. *Host* can be entered as a node name or network address. The node name is specified in the Intergraph clearinghouse (see *clh*(1)). A network address has the form [*xxxxxx.*]*aa-bb-cc-dd-ee-ff*, where *xxxxxx* is an optional Local Area Network (LAN) number, and *aa-bb-cc-dd-ee-ff* is an Ethernet address.

visit searches the login directory for a **.rloginrc** file that can be used to specify default options and a simple *chat* script.

The following sections describe the options available from the command line. They can also be entered using the *visit* **connect** command. The **-p** option defines the protocol to be used for the current session. The valid *protocols* are **xt** and **vtp**. The default *protocol* is **xt**.

The following three sections describe options available to *visit* on the command line and as options to the *visit* **connect** command.

General Options

The following options can be used for all protocols:

-e *chars* Specify an escape sequence to access interactive mode. *Chars* specifies a sequence of characters, where ^ represents <CONTROL> and \^ represents ^. The default sequence is <CONTROL>-Y <CONTROL>-Y. To prevent escaping to interactive mode, a null escape sequence may be entered (such as **-e** ""). This is useful for captive accounts.

-f *logfile* Specify a log file on the local machine. If the specified file does not exist, it will be created. If the file exists, it will be overwritten.

-i *time* Specify the *time* (in 1/60-second intervals) that *visit* will check for terminal input. The default is 5.

-n Ignore the login script **.rloginrc**. However, default *visit* **connect** options in the login script are not ignored.

-q Prevent the display of certain *visit* messages. These messages include the XON/XOFF message that appears during a *visit* **connect** and the termination message. This option is useful for cosmetic purposes in shell scripts that invoke *visit*.

-r Send a <RETURN> to the remote *host* after a connection is established. This option can be used if the remote *host* does not automatically prompt the user to log in.

-s *baudrate* Set the terminal baud rate for the *visit* session.

-w Stop *visit* from waiting for input from a device attached to an auxiliary port.

-x Cause the local system to interpret XON/XOFF (<CONTROL>-S/<CONTROL>-Q) flow control. This is the default if the local terminal is set to **ixon** (see *stty*(1)).

-y Allow XON/XOFF (<CONTROL>-S/<CONTROL>-Q) to be passed to the remote *host* instead of being interpreted by the local system. This option is useful when running programs on the remote *host* that need to interpret the XON/XOFF character sequences. For example, emacs uses <CONTROL>-S as a command. The **-y** option is the opposite of the **-x** option. This is the default if the local terminal is set to **-ixon** (see *stty*(1)).

-? Display a usage message and exit.

XT Protocol Options

-o Prevent the connection to the remote *host* from being terminated on logout. To disconnect or exit from the remote system, enter the escape sequence in interactive mode.

-t *device* If a remote *host* is running CLIX, *device* is a device number of the remote terminal device. (A *device* of 5 would correspond to the terminal device **/dev/ttn05**.) If the device is preceded by a +, a *getty*(1M) will not be started.

If a remote *host* is an Intergraph VAX/VMS system, an XT device name becomes associated with the logical name *device*.

VTP Protocol Options

-c Configure a Communications Server. Only the super-user can execute this option. No other options should be specified.

-l *port#* Specify the port on a Communications Server for the connection. See the *Intergraph XNS/VTP Administrator's Guide* for a discussion of port and rotary numbers.

-j *address* Specify an X.25 address to access a host on a Public Data Network (PDN) through an XNS/X.25 gateway. If the X.25 host is connected directly to an XNS/X.25 Gateway, the X.25 address need not be specified.

Interactive Commands

? [*command*]	Display help information for the specified *command*. If no *command* is given, list all available commands.
! [*local command*]	Execute a command on the local host. Specifying ! alone will start a shell process.
connect [*option* ...] *host*	Connect to the specified *host*. All options described above can be used with the *visit* **connect** command. Each *visit* **connect** establishes a session (maximum of eight sessions). Sessions are numbered starting with 0.
disconnect *session*	Disconnect *session*.
exit	Terminate all connections and exit.
help [*command*]	Synonym for the ? command.
quit	Synonym for the **exit** command.
resume *session#*	Resume *session#*.
show_sessions	List all current sessions.
stop_log *session#*	Stop logging *session#*. Logging must have been turned on by the **-f** option.
unstop_log *session#*	Resume logging *session#*.
version	Display the *visit* release date.

.rloginrc File Commands

The following commands are available in the **.rloginrc** startup file. The first line of a file may contain a **c** followed by a list of options. The following lines may contain *chat* scripts for different remote hosts. Each *chat* script begins with a line starting with a **!**.

connect [*option* ...]	List default *visit* **connect** options. These options can be overridden on the command line. This must be the first line of the **.rloginrc** file if it is included.
! [*host*]	Start a new *chat* script. If *host* is specified, this *chat* script will be used every time a session is started with *host*. *host* may be specified as a node name or network address. To override this *chat* script, use the **-n** option. If an argument is not specified, this *chat* script will be the default *chat* script.
output *string*	Output *string* to the session once it is connected. *String* is a sequence of characters, where ^ represents <CONTROL> and \^ represents ^.
input *timeout string*	Wait for *host* to print *string*. *String* is a sequence of characters, where ^ represents <CONTROL> and \^ represents ^. The *timeout* is specified in seconds. If

string is not received within *timeout* seconds, *visit* will ignore the rest of the *chat* script.

EXAMPLES

An example **.rloginrc** would appear as follows:

```
connect -y -e "" -f logfile.dat
! is200
          input 5 login:
          output joe^M
          input 2 word:
          output abc123^M
          input 60 $
          output who^M
```

The first line specifies the default *visit* **connect** options to be used anytime a session is started. The rest of the script is an example of a *chat* script to be used when a connection is made to "is200."

When a connection is made to "is200," *visit* sets the default options "-y -e "" -f logfile.dat" and then waits a maximum of five seconds for the string "login:." When this string is received, the string "joe<RETURN>" is sent to the remote system. *visit* then waits a maximum of two seconds for "word:," the last portion of "password:." *visit* sends the password "abc123<RETURN>," waits a maximum of 60 seconds for the "$" prompt, and finally sends "who<RETURN>" to the remote system and returns control to the user.

SEE ALSO

Intergraph Network Core User's Guide.
XNS/VTP Administrator's Guide.

NAME

vmsbackup - read a VMS backup tape

SYNOPSIS

vmsbackup [**-tx**] [**-cdevw**] [**-s** *setnum*] [**-n** *setname*] [**-f** *tapefile*] [*name* ...]

DESCRIPTION

vmsbackup reads a VMS-generated backup tape and writes the files to a CLIX disk. The default operation of the program is to extract every file from the tape and write it to disk. The default may be modified by the following options:

-c Use complete file names including the version number. A colon and the octal version number will be appended to all file names. This option is useful only when multiple versions of the same file are on a single tape or when a file with the same name exists in the destination directory. By default, version numbers are ignored.

-d Use the directory structure from VMS.

-e Process all file name extensions. Since this program is mainly intended to move source code and possibly data from a VMS system to a CLIX system, the default is to ignore all files whose file name extensions specify system-dependent data. The file types that will be ignored unless the **-e** option is specified are as follows:

exe	VMS executable file
lib	VMS object library file
obj	RSX object file
odl	RSX overlay description file
olb	RSX object library file
pmd	RSX post-mortem dump file
stb	RSX task symbol table file
sys	RSX bootable system file
tsk	RSX executable task file

-f Use the next argument in the command line as the tape device to be used rather than the default **/dev/rmt/0m**.

-s *setnumber* Process only the given saveset number.

-n *setname* Process only savesets on the tape whose names match the *setname* argument. Pattern matching in the manner of *sh*(1) is attempted using the meta-characters *****, **?**, **!**, **[**, and **]**.

-t *tapefile* Produce a table of contents (a directory listing) on the standard output of the files on tape.

-v Set verbose mode. The verbose option will cause the names of the files being read from tape to be written to standard output.

-w Query the user for file disposition. *vmsbackup* prints the message "extract *file-name* [ny]" and waits for user confirmation that the file is to be extracted. If a word beginning with **y** is given, the file is copied to the file system. Any other input is interpreted as no.

-x Extract the named files from the tape. The optional *name* argument specifies one or more file names to be searched for on the tape. Pattern matching in the manner of *sh*(1) is attempted using the meta-characters *****, **?**, **!**, **[**, and **]**. Only files with matching names are processed.

FILES

/dev/rmt/0m	default tape device

CAVEATS

The file name match uses the complete VMS file names.

NAME

watcher - CRM utility for monitoring system calls and faults

SYNOPSIS

/usr/ip32/crm/watcher *event-options* [**-a**] [**-o** *output-file*] *input-option*

DESCRIPTION

If a summary of the system calls was selected to be displayed, the calls will be displayed when *watcher* is exited. Otherwise, a scrolling list of the system calls and faults being encountered by the specified process will be displayed.

The following *event-options* are available:

-f *fault-options*	Enable monitoring of system faults. The **ALL** option will provide monitoring of all system faults. Other *fault-options* that can be defined are **demand**, **swap**, **cache**, **file**, **cw**, and **steal**.
-s *system-call-types*	Enable monitoring of system calls. The **ALL** option enables all system calls to be monitored. Other system calls are program-specific and are therefore user-definable.

The following options are available:

-a	Translate addresses. If the program was compiled to include debugger symbols (such as to be used by *dbg*(1)), *watcher* can read these symbols and provide more logical values for the program counter (PC).
-o *output-file*	Specify an *output-file* for raw data to be stored in. A - can be used to direct output to **stdout**.

The following *input-options* are available:

-i *input-file*	Read the data from *input-file*. *Input-file* must have been created as an *output-file* using the **-o** option. A - for *input-file* reads input from **stdin**.
-p *pid*	Specify the ID number of the process to monitor (PID). The user can key in **ps -e** at the system prompt to determine the PID of a process already running.
-n *process-name*	Specify the name of the process to monitor. The user can key in **ps -e** at the system prompt to determine the name of a process already running.
-e *command* [*arg* ...]	Allow the user to run, provide arguments for, and monitor a program.

A brief explanation of the *watcher* System Faults fields follows:

System Fault	Displays the occurrence of a page fault and the fault type (such as demand, swap, cache).
PC (program counter)	Displays the address of the program instruction that took the fault.
Virtual address	Displays the address that was accessed to cause the fault.

A brief explanation of the *watcher* System Calls fields follows:

PC (program counter)	Displays the address of the program instruction that issued the system call.
arg0, arg1 ... arg*n*	Displays any arguments of the system call.
completion status	Displays the success or failure of a system call or provides data about the call. For instance, a *write*(2) would display a value in this field to indicate the number of bytes that were written during the call.

EXAMPLES

A sample of system call and system fault fields are displayed in the Profiler as follows:

```
System Faults: DEMAND
        PC: 0x00004400
        Virtual address: 0x0040157d

OPEN system call
        PC:   0xff804e62
        arg0:  647773 7361702F 6374652F  /etc/passwd
        arg1:         00000000  ....
         completion status: 1
```

SEE ALSO

crm(1), write(2).

WARNINGS

Sending raw data to a file can create a very large file.

NAME

ypcat - print values in a YP database

SYNOPSIS

ypcat [**-k**] [**-t**] [**-d** *domain-name*] *mname*
ypcat -x

DESCRIPTION

ypcat prints values in a Yellow Pages (YP) map specified by *mname*, which may be either a *mapname* or a map nickname. Since *ypcat* uses the YP network services, no YP server is specified.

To look at the network-wide password database, **passwd.byname**, (with the nickname **passwd**) key in:

ypcat passwd

The following options are available:

-k	Display the keys for maps in which the values are null or the key is not part of the value. (None of the maps derived from files that have an ASCII version in **/etc** fall in this class.)
-t	Inhibit translation of *mname* to *mapname*. For example, **ypcat -t passwd** fails because no map is named **passwd**; whereas, **ypcat passwd** is translated to **ypcat passwd.byname**.
-d *domain-name*	Specify a domain other than the default. The default domain is returned by *domname*(1).
-x	Display the map nickname table. This lists the nicknames (*mnames*) with which the command is familiar and indicates the *mapname* associated with each nickname.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of the YP.

SEE ALSO

ypfiles(4), ypmatch(1), domname(1).
ypserv(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

ypmatch - print the value of one or more keys from a YP map

SYNOPSIS

ypmatch [**-d** *domain*] [**-k**] [**-t**] *key* ... *mname*
ypmatch -x

DESCRIPTION

ypmatch prints the values associated with one or more keys from the Yellow Pages (YP) map (database) specified by a *mname*, which may be either a *mapname* or a map nickname.

Multiple *keys* can be specified; the same map is searched for all. The *keys* must be exact values in capitalization and length. No pattern matching is available. If a *key* is not matched, a diagnostic message is produced.

The following options are available:

-d Specify a domain other than the default.

-k Before printing the value of a *key*, print the *key* itself followed by a colon (":"). This is useful only if the *keys* are not duplicated in the values, or so many *keys* have been specified that the output could be confusing.

-t Inhibit translation of nickname to mapname. For example, **ypmatch -t zippy passwd** fails because no map is named **passwd**, while **ypmatch zippy passwd** is translated to **ypmatch zippy passwd.byname**.

-x Display the map nickname table. This lists the nicknames (*mnames*) with which the command is familiar and indicates the *mapname* associated with each nickname.

SEE ALSO

ypfiles(4), ypcat(1).

NAME

yppasswd - change login password in YP

SYNOPSIS

yppasswd [*name*]

DESCRIPTION

yppasswd changes or installs a password associated with the user *name* (login name default) in the Yellow Pages (YP). The YP password may be different from the one on the local machine.

yppasswd prompts for the old YP password and then for the new one. The user must supply both. The new password must be typed twice to avoid mistakes. New passwords must have at least four characters if they use a sufficiently-rich alphabet (uppercase, lowercase, and nonalphabetic characters) or at least six characters if monocase (all uppercase or all lowercase).

Only the name owner or super-user may change a password; in either case the old password must be supplied.

SEE ALSO

ypfiles(4).
yppasswdd(1M) in the *CLIX System Administrator's Reference Manual.*
passwd(1) in the *UNIX System V User's Reference Manual.*

BUGS

The update protocol passes all information to the server in one Remote Procedure Call without looking at it. Thus, if the old password is typed in incorrectly, notification will not be sent until after the new password has been entered.

System Calls (2)

NAME

intro - introduction to system calls and error numbers

SYNOPSIS

#include <errno.h>

DESCRIPTION

This section describes all system calls. Certain major collections are identified by a letter after the section number:

(2B) Certain 4.3 Berkeley Softtware Distribution (BSD) functionality was added to CLIX through additional system calls. The system calls can be accessed with the library *libbsd*. They are not automatically loaded as needed by the C compiler, *cc*(1). However, the link editor, *ld*(1), searches this library under the **-lbsd** option.

(2I) These system calls are CLIX-specific calls. The system calls can be accessed with the Intergraph Library *libix*. They are not automatically loaded as needed by the C compiler, *cc*(1); however, the link editor, *ld*(1), searches this library under the **-lix** option.

Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value. This is almost always -1 or the null pointer; the individual descriptions specify the details. An error number is also available in the external variable *errno*. *Errno* is not cleared on successful calls, so it should be tested only after an error is indicated.

Each system call description attempts to list all possible error numbers. The following is a complete list of the error numbers and their names as defined in **<errno.h>**.

1 EPERM Not owner
: Typically this error indicates an attempt to modify a file forbidden except to its owner or super-user. It is also returned if an ordinary user attempts an action allowed only to the super-user.

2 ENOENT No such file or directory
: This error occurs when a file name is specified and the file should exist but does not, or when one of the directories in a path name does not exist.

3 ESRCH No such process
: No process can be found that corresponds to the process specified by *pid* in *kill*(2) or *ptrace*(2).

4 EINTR Interrupted system call
: An asynchronous signal (such as interrupt or quit), that the user elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

5 EIO I/O error
: Some physical I/O error occurred. This error may in some cases occur on a call following the one to which it actually applies.

6 ENXIO No such device or address
: I/O on a special file refers to a subdevice that does not exist or is beyond the limits of the device. It may also occur when, for example, a tape drive is not online or no disk pack is loaded on a drive.

7 E2BIG Arg list too long
: An argument list longer than 5,120 bytes is presented to a member of the *exec*(2) family.

8 ENOEXEC Exec format error
: Execution of a file that, although it has the appropriate permissions, does not start with a valid magic number (see *a.out*(4)) is requested.

9 EBADF Bad file number
: Either a file descriptor refers to no open file or a *read*(2) (respectively, *write*(2)) request is made to a file that is open only for writing (respectively, reading).

10 ECHILD No child processes
: A *wait*(2) was executed by a process that had no existing or unwaited-for child processes.

11 EAGAIN No more processes
: A *fork*(2) failed because the system's process table is full or the user is not allowed to create any more processes. Or a system call failed because of insufficient memory or swap space.

12 ENOMEM Not enough space
: During an *exec*(2), *brk*(2), or *sbrk*(2), a program asks for more space than the system is able to supply. This may not be a temporary condition; the maximum space size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers, or if there is not enough swap space during a *fork*(2). This error occurring on a resource associated with Remote File Sharing (RFS) indicates a memory depletion that may be temporary, depending on system activity at the time the call was invoked.

13 EACCES Permission denied
: An attempt was made to access a file in a way forbidden by the protection system.

14 EFAULT Bad address
: The system encountered a hardware fault in attempting to use an argument of a system call.

15 ENOTBLK Block device required
: A nonblock file was mentioned where a block device was required (as in *mount*(2)).

16 EBUSY Device or resource busy
: An attempt was made to mount a device already mounted or to dismount a device that has an active file (open file, current directory, mounted-on file, active text segment). It will also occur if an attempt is made to enable accounting when it is already enabled. The device or resource is currently unavailable.

17 EEXIST File exists
: An existing file was mentioned in an inappropriate context (such as *link*(2)).

18 EXDEV Cross-device link
: A link to a file on another device was attempted.

19 ENODEV No such device
: An attempt was made to apply an inappropriate system call to a device (such as to read a write-only device).

20 ENOTDIR Not a directory
: A nondirectory was specified where a directory is required (such as in a path prefix or as an argument to *chdir*(2)).

21 EISDIR Is a directory
: An attempt was made to write on a directory.

22 EINVAL Invalid argument
: Some invalid argument (such as dismounting a nonmounted device; mentioning an undefined signal in *signal*(2) or *kill*(2); or reading or writing a file for which *lseek*(2) has generated a negative pointer). The error is Also set by the math functions described in the (3M) entries of this manual.

23 ENFILE File table overflow
: The system file table is full, and temporarily no more *opens* can be accepted.

24 EMFILE Too many open files
: No process may have more than NOFILES (default 128) descriptors open at a time.

25 ENOTTY Not a character device (or) Not a typewriter
: An attempt was made to *ioctl*(2) a file that is not a special character device.

26 ETXTBSY Text file busy
: An attempt was made to execute a pure-procedure program currently open for writing. Also, an attempt to open for writing or to remove a pure-procedure program being executed.

27 EFBIG File too large
: The size of a file exceeded the maximum file size or ULIMIT (see *ulimit*(2)).

28 ENOSPC No space left on device
: During a *write*(2) to an ordinary file, no free space is left on the

device. In *fcntl*(2), the setting or removing of record locks on a file cannot be accomplished because no more record entries remain on the system.

29 ESPIPE Illegal seek
: An *lseek*(2) was issued to a pipe.

30 EROFS Read-only file system
: An attempt to modify a file or directory was made on a device mounted read-only.

31 EMLINK Too many links
: An attempt was made to make more than the maximum number of links (1000) to a file.

32 EPIPE Broken pipe
: A write on a pipe for which no process to read the data exists. This condition normally generates a signal; the error is returned if the signal is ignored.

33 EDOM Math argument
: The argument of a function in the math package (3M) is out of the function's domain.

34 ERANGE Result too large
: The value of a function in the math package (3M) is not representable within machine precision.

35 ENOMSG No message of desired type
: An attempt was made to receive a message of a type that does not exist on the specified message queue (see *msgop*(2)).

36 EIDRM Identifier removed
: This error is returned to processes that resume execution due to the removal of an identifier from the file system's name space (see *msgctl*(2), *semctl*(2), and *shmctl*(2)).

37-44 Reserved numbers

45 EDEADLK Deadlock
: A deadlock situation was detected and avoided. This error pertains to file and record locking.

46 ENOLCK No lock
: In *fcntl*(2), setting or removing record locks on a file cannot be accomplished because no more record entries remain on the system.

60 ENOSTR Not a stream
: A *putmsg*(2) or *getmsg*(2) system call was attempted on a file descriptor that is not a STREAMS device.

62 ETIME Stream ioctl timeout
: The timer set for a STREAMS *ioctl*(2) call expired. The cause of this error is device specific and could indicate a hardware or software failure or a timeout value that is too short for the specific operation.

The status of the *ioctl*(2) operation is indeterminate.

63 ENOSR No stream resources
During a STREAMS *open*(2), either no STREAMS queues or no STREAMS head data structures were available.

64 ENONET Machine is not on the network
This error is Remote File Sharing (RFS) specific. It occurs when users try to advertise, unadvertise, mount, or unmount remote resources when the machine did no do the proper startup to connect to the network.

65 ENOPKG No package
This error occurs when users attempt to use a system call from a package that is not installed.

66 EREMOTE Resource is remote
This error is RFS specific. It occurs when users try to advertise a resource that is not on the local machine or try to mount/unmount a device (or path name) that is on a remote machine.

67 ENOLINK Virtual circuit is gone
This error is RFS specific. It occurs when the link (virtual circuit) connecting to a remote machine is gone.

68 EADV Advertise error
This error is RFS-specific. It occurs when users try to advertise a resource that has been advertised, try to stop the RFS while resources are still advertised, or try to force an unmount on a resource when it is still advertised.

69 ESRMNT Srmount error
This error is RFS-specific. It occurs when users try to stop RFS while resources are still mounted by remote machines.

70 ECOMM Communication error
This error is RFS-specific. It occurs when users try to send messages to remote machines, but no virtual circuit can be found.

71 EPROTO Protocol error
Some protocol error occurred. This error is device specific, but is generally not related to a hardware failure.

74 EMULTIHOP Multihop attempted
This error is RFS-specific. It occurs when users try to access remote resources that are not directly accessible.

77 EBADMSG Bad message
During a *read*(2), *getmsg*(2), or *ioctl*(2) I_RECVFD system call to a STREAMS device, something that cannot be processed has come to the head of the queue. What it is depends on the system call:

read(2) - Control information or a passed file descriptor.
getmsg(2) - Passed file descriptor.
ioctl(2) - Control or data information.

83 ELIBACC Cannot access a needed shared library
: Tried to *exec*(2) an *a.out*(4) that requires a shared library (to be linked in) and the shared library does not exist or the user does not have permission to use it.

84 ELIBBAD Accessing a corrupted shared library
: Tried to *exec*(2) an *a.out*(4) that requires a shared library (to be linked in) and *exec*(2) could not load the shared library. The shared library is probably corrupted.

85 ELIBSCN .lib section in *a.out*(4) corrupted
: Tried to *exec*(2) an *a.out*(4) that requires a shared library (to be linked in) and erroneous data was in the .lib section of the *a.out*(4). The .lib section tells *exec*(2) the shared libraries needed. The *a.out*(4) is probably corrupted.

86 ELIBMAX Attempting to link in more shared libraries than system limit
: Tried to *exec*(2) an *a.out*(4) that requires more shared libraries (to be linked in) than allowed on the current system configuration.

87 ELIBEXEC Cannot exec a shared library directly
: Tried to *exec*(2) a shared library directly. This is not allowed.

90 EWOULDBLOCK Operation would block
: An operation that would cause a process to block was attempted on an object in nonblocking mode (see *fcntl*(2)).

91 EINPROGRESS Operation now in progress
: An operation that takes a long time to complete (such as a *connect*(2B)) was attempted on a nonblocking object (see *fcntl*(2)).

92 EALREADY Operation already in progress
: An operation was attempted on a nonblocking object that had an operation in progress.

93 ENOTSOCK Socket operation on nonsocket
: Self-explanatory.

94 EDESTADDRREQ Destination address required
: A required address was omitted from an operation on a socket.

95 EMSGSIZE Message too long
: A message sent on a socket was larger than the internal message buffer or some other network limit.

96 EPROTOTYPE Protocol wrong type for socket
: A protocol that does not support the semantics of the socket type requested was specified.

97 EPROTONOSUPPORT Protocol not supported
: The protocol was not configured in the system or no implementation exists for it.

98 ESOCKTNOSUPPORT Socket type not supported
: The support for the socket type was not configured in the system or

no implementation exists for it.

99 EOPNOTSUPP Operation not supported on socket
Self-explanatory.

100 EPFNOSUPPORT Protocol family not supported
The protocol family was not configured in the system or no implementation exists for it.

101 EAFNOSUPPORT Address family not supported by protocol family
An address incompatible with the requested protocol was used.

102 EADDRINUSE Address already in use
Only one use of each address is normally permitted.

103 EADDRNOTAVAIL Can't assign requested address
This normally results from an attempt to create a socket with an address not on this machine.

104 ENETDOWN Network is down
A socket operation encountered a dead network.

105 ENETUNREACH Network is unreachable
A socket operation was attempted to an unreachable network.

106 ENETRESET Network dropped connection on reset
The host connected to crashed and rebooted.

107 ECONNABORTED Software caused connection abort
A connection abort was caused internal to the host machine.

108 ECONNRESET Connection reset be peer
A connection was forcibly closed by a peer. This normally results from a loss of the connection on the remote socket due to a timeout or a reboot.

109 ENOBUFS No buffer space available
An operation on a socket was not performed because the system lacked sufficient buffer space or because a queue was full.

110 EISCONN Socket is already connected
A *connect*(2B) request was made on an already connected socket or a *sendto*(2B) request on a connected socket specified a destination when it was already connected.

111 ENOTCONN Socket is not connected
An request to send or receive data was disallowed because the socket is not connected and (when sending on a datagram socket) no address was supplied.

112 ESHUTDOWN Can't send after socket shutdown
A request to send data was disallowed because the socket was shut down with a previous *shutdown*(2B) call.

114 ETIMEDOUT Connection timed out
A socket operation timed out. The timeout period depends on the communication protocol.

115 ECONNREFUSED Connection refused

No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host.

116 EHOSTDOWN Host is down

A socket operation failed because the destination host was down.

117 EHOSTUNREACH No route to host

A socket operation was attempted to an unreachable host.

118 ENOPROTOOPT Protocol not available

A bad option or level was specified in a *getsockopt*(2B) or *setsockopt*(2B) call.

Definitions

Process ID

Each active process in the system is uniquely identified by a positive integer called a process ID. The range of this ID is from 1 to 30,000.

Parent Process ID

A new process is created by a currently active process (see *fork*(2)). The parent process ID of a process is the **process ID** of its creator.

Process Group ID

Each active process is a member of a process group identified by a positive integer called the process group ID. This ID is the **process ID** of the group leader. This grouping permits the signaling of related processes (see *kill*(2)).

Tty Group ID

Each active process can be a member of a terminal group identified by a positive integer called the tty group ID. This grouping is used to terminate a group of related processes when one of the processes in the group is terminated (see *exit*(2) and *signal*(2)).

Real User ID and Real Group ID

Each user allowed on the system is identified by a positive integer (0 to 65535) called a real user ID.

Each user is also a member of a group. The group is identified by a positive integer called the real group ID.

An active process has a real user ID and real group ID that are set to the real user ID and real group ID, respectively, of the user responsible for the process creation.

Effective User ID and Effective Group ID

An active process has an effective user ID and an effective group ID used to determine **file access permissions**. The effective user ID and effective group ID are equal to the process's **real user ID** and **real group ID**, respectively unless the process or one of its ancestors evolved from a file that had the set-user-ID bit or set-group ID bit set (see *exec*(2)).

Super-user

A process is recognized as a super-user process and is granted special privileges, such as immunity from file permissions, if its **effective user ID** is 0.

Special Processes

The processes with a **process ID** of 0 and a **process ID** of 1 are special processes and are referred to as *proc0* and *proc1*.

Proc0 is the scheduler. *Proc1* is the initialization process (*init*). *Proc1* is the ancestor of every other process in the system and is used to control the process structure.

File Descriptor

A file descriptor is a small integer used to perform I/O on a file. The value of a file descriptor is from 0 to (NOFILES - 1). A process may have no more than NOFILES file descriptors open simultaneously. A file descriptor is returned by system calls such as *open*(2) or *pipe*(2). The file descriptor is used as an argument by calls such as *read*(2), *write*(2), *ioctl*(2), and *close*(2).

File Name

Names consisting of 1 to 14 characters may be used to name an ordinary file, special file, or **directory**.

These characters may be selected from the set of all character values excluding \0 (null) and the ASCII code for / (slash).

It is generally unwise to use *, ?, [, or] as part of file names because of the special meaning attached to these characters by the shell (see *sh*(1)). Although permitted, using unprintable characters in file names should be avoided.

Path Name and Path Prefix

A path name is a null-terminated character string starting with an optional slash (/), followed by zero or more **directory** names separated by slashes, and optionally followed by a file name.

If a path name begins with a slash, the path search begins at the **root directory**. Otherwise, the search begins from the **current working directory**.

A slash by itself names the **root directory**.

Unless specifically stated otherwise, the null path name is treated as if it named a nonexistent file.

Directory

Directory entries are called links. By convention, a directory contains at least two links, . and .., referred to as dot and dot-dot, respectively. Dot is the directory itself and dot-dot is its parent directory.

Root Directory and Current Working Directory

Each process has associated with it a concept of a root directory and a current working directory for resolving path name searches. The root directory of a process need not be the root directory of the root file system.

File Access Permissions

Read, write, and execute/search permissions on a file are granted to a process if one or more of the following is true:

The **effective user ID** of the process is **super-user.**

The **effective user ID** of the process matches the **user ID** of the file owner and the appropriate access bit of the "owner" portion (0700) of the file mode is set.

The **effective user ID** of the process does not match the **user ID** of the file owner, the **effective group ID** of the process matches the group of the file, and the appropriate access bit of the file mode's "group" portion (0070) is set.

The **effective user ID** of the process does not match the **user ID** of the file owner, the **effective group ID** of the process does not match the **group ID** of the file, and the appropriate access bit of the file mode's "other" portion (0007) is set.

Otherwise, the corresponding permissions are denied.

Message Queue Identifier

A message queue identifier (msqid) is a unique positive integer created by a *msgget*(2) system call. Each msqid has an associated message queue and data structure. The data structure is referred to as *msqid_ds* and contains the following members:

```
struct   ipc_perm msg_perm;
struct   msg *msg_first;
struct   msg *msg_last;
ushort   msg_cbytes;
ushort   msg_qnum;
ushort   msg_qbytes;
ushort   msg_lspid;
ushort   msg_lrpid;
time_t   msg_stime;
time_t   msg_rtime;
time_t   msg_ctime;
```

msg_perm

An ipc_perm structure that specifies the message operation permission (see below). This structure includes the following members:

```
ushort   cuid;          /* creator user ID */
ushort   cgid;          /* creator group ID */
ushort   uid;           /* user ID */
ushort   gid;           /* group ID */
ushort  mode;        /* r/w permission */
ushort  seq;         /* slot usage sequence # */
key_t   key;         /* key */
```

msg *msg_first
A pointer to the first message on the queue.

msg *msg_last
A pointer to the last message on the queue.

msg_cbytes
The current number of bytes on the queue.

msg_qnum
The number of messages currently on the queue.

msg_qbytes
The maximum number of bytes allowed on the queue.

msg_lspid
The **process ID** of the last process that performed a *msgsnd* operation.

msg_lrpid
The **process ID** of the last process that performed a *msgrcv* operation.

msg_stime
The time of the last *msgsnd* operation.

msg_rtime
The time of the last *msgrcv* operation

msg_ctime
The time of the last *msgctl*(2) operation that changed a member of the above structure.

Message Operation Permissions

In the *msgop*(2) and *msgctl*(2) system call descriptions, the permission required for an operation is given as {*token*}, where *token* is the type of permission needed, interpreted as follows:

00400	Read by user
00200	Write by user
00040	Read by group
00020	Write by group
00004	Read by others
00002	Write by others

Read and write permissions on a *msqid* are granted to a process if one or more of the following is true:

The **effective user ID** of the process is **super-user**.

The **effective user ID** of the process matches **msg_perm.cuid** or **msg_perm.uid** in the data structure associated with *msqid* and the appropriate bit of the "user" portion (0600) of **msg_perm.mode** is set.

The **effective group ID** of the process matches **msg_perm.cgid** or **msg_perm.gid** and the appropriate bit of the "group" portion (060) of **msg_perm.mode** is set.

The appropriate bit of the "other" portion (006) of **msg_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

Semaphore Identifier

A semaphore identifier (semid) is a unique positive integer created by a *semget*(2) system call. Each semid has a set of semaphores and a data structure associated with it. The data structure is referred to as *semid_ds* and contains the following members:

```
struct   ipc_perm sem_perm;/* operation permission struct */
struct   sem *sem_base;      /* ptr to first semaphore in set */
ushort   sem_nsems;          /* number of sems in set */
time_t   sem_otime;          /* last operation time */
time_t   sem_ctime;          /* last change time */
                             /* Times measured in secs since */
                             /* 00:00:00 GMT, Jan. 1, 1970 */
```

sem_perm

An ipc_perm structure that specifies the **semaphore operation permission**. This structure includes the following members:

```
ushort   uid;          /* user ID */
ushort   gid;          /* group ID */
ushort   cuid;         /* creator user ID */
ushort   cgid;         /* creator group ID */
ushort   mode;         /* r/a permission */
ushort   seq;          /* slot usage sequence number */
key_t    key;          /* key */
```

sem_nsems

Equal to the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as a *sem_num*. Sem_num values run sequentially from 0 to the value of sem_nsems minus 1.

sem_otime

The time of the last *semop*(2) operation.

sem_ctime

The time of the last *semctl*(2) operation that changed a member of the above structure.

A semaphore is a data structure called *sem* that contains the following members:

```
ushort  semval;     /* semaphore value */
short   sempid;     /* pid of last operation  */
ushort  semncnt;    /* # awaiting semval > cval */
ushort  semzcnt;    /* # awaiting semval = 0 */
```

semval
A non-negative integer that is the actual value of the semaphore.

sempid
Equal to the **process ID** of the last process that performed a semaphore operation on this semaphore.

semncnt
A count of the number of processes currently suspended and awaiting this semaphore's semval to become greater than its current value.

semzcnt
A count of the number of processes currently suspended and awaiting this semaphore's semval to become zero.

Semaphore Operation Permissions

In the *semop*(2) and *semctl*(2) system call descriptions, the permission required for an operation is given as {*token*}, where *token* is the type of permission needed, interpreted as follows:

00400	read by user
00200	alter by user
00040	read by group
00020	alter by group
00004	read by others
00002	alter by others

Read and alter permissions on a **semid** are granted to a process if one or more of the following is true:

The **effective user ID** of the process is **super-user**.

The **effective user** ID of the process matches **sem_perm.cuid** or **sem_perm.uid** in the data structure associated with **semid** and the appropriate bit of the "user" portion (0600) of **sem_perm.mode** is set.

The **effective group ID** of the process matches **sem_perm.cgid** or **sem_perm.gid** and the appropriate bit of the "group" portion (060) of **sem_perm.mode** is set.

The appropriate bit of the "other" portion (006) of **sem_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

Shared Memory Identifier

A shared memory identifier (shmid) is a unique positive integer created by a *shmget*(2) system call. Each shmid has a segment of memory (referred to as a shared memory segment) and an associated data structure. (These shared memory segments must be explicitly removed by the user after the last reference to them is removed.) The data structure is referred to as *shmid_ds* and contains the following members:

```
struct   ipc_perm shm_perm;/* operation permission struct */
int      shm_segsz;         /* size of segment */
struct   region *shm_reg;   /*ptr to region structure */
char     pad[4];            /* for swap compatibility */
ushort   shm_lpid;          /* pid of last operation */
ushort   shm_cpid;          /* creator pid */
ushort   shm_nattch;        /* number of current attaches */
ushort   shm_cnattch;       /* used only for shminfo */
time_t   shm_atime;         /* last attach time */
time_t   shm_dtime;         /* last detach time */
time_t   shm_ctime;         /* last change time */
                            /* Times measured in secs since */
                            /* 00:00:00 GMT, Jan. 1, 1970 */
```

shm_perm

An *ipc_perm* structure that specifies the **shared memory operation permission**. This structure includes the following members:

```
ushort   cuid;       /* creator user ID */
ushort   cgid;       /* creator group ID */
ushort   uid;        /* user ID */
ushort   gid;        /* group ID */
ushort   mode;       /* r/w permission */
ushort   seq;        /* slot usage sequence # */
key_t    key;        /* key */
```

shm_segsz

The size of the shared memory segment in bytes.

shm_cpid

The **process ID** of the process that created the shared memory identifier.

shm_lpid

The **process ID** of the last process that performed a *shmop*(2) operation.

shm_nattch

The number of processes that currently have this segment attached.

shm_atime
The time of the last *shmat*(2) operation.

shm_dtime
The time of the last *shmdt*(2) operation.

shm_ctime
The time of the last *shmctl*(2) operation that changed a member of the above structure.

Shared Memory Operation Permissions
In the *shmop*(2) and *shmctl*(2) system call descriptions, the permission required for an operation is given as {*token*}, where *token* is the type of permission needed, interpreted as follows:

00400	read by user
00200	write by user
00040	read by group
00020	write by group
00004	read by others
00002	write by others

Read and write permissions on a *shmid* are granted to a process if one or more of the following is true:

The **effective user ID** of the process is **super-user**.

The **effective user ID** of the process matches *shm_perm.cuid* or *shm_perm.uid* in the data structure associated with *shmid* and the appropriate bit of the "user" portion (0600) of *shm_perm.mode* is set.

The **effective group ID** of the process matches *shm_perm.cgid* or *shm_perm.gid* and the appropriate bit of the "group" portion (060) of *shm_perm.mode* is set.

The appropriate bit of the "other" portion (06) of *shm_perm.mode* is set.

Otherwise, the corresponding permissions are denied.

STREAMS
A set of kernel mechanisms that support the development of network services and data communication **drivers**. It defines interface standards for character input/output within the kernel and between the kernel and user-level processes. The STREAMS mechanism is composed of utility routines, kernel facilities, and a set of data structures.

Stream
A full-duplex data path within the kernel between a user

process and driver routines. The primary components are a **stream head**, a **driver**, and zero or more **modules** between the **stream head** and **driver**. A stream is analogous to a shell pipeline except that data flow and processing are bidirectional.

Stream Head
The end of the **stream** that provides the interface between the **stream** and a user process. The principle functions of the stream head are processing **STREAMS**-related system calls and passing data and information between a user process and the **stream**.

Driver
The interface between peripheral hardware and the **stream**. A driver can also be a pseudo-driver, such as a **multiplexor** or log driver (see *log*(7)), that is not associated with a hardware device.

Module
An entity containing processing routines for input and output data. It always exists in the middle of a **stream** between the stream's head and a **driver**. A **module** is the **STREAMS**' counterpart to the commands in a shell pipeline except that a module contains a pair of functions that allow independent bidirectional (**downstream** and **upstream**) data flow and processing.

Downstream
In a **stream**, the direction from **stream head** to **driver**.

Upstream
In a **stream**, the direction from **driver** to **stream head**.

Message
In a **stream**, one or more blocks of data or information with associated **STREAMS** control structures. **Messages** can be of several defined types, that identify the **message** contents. **Messages** are the only means of transferring data and communicating within a **stream**.

Message Queue
In a **stream**, a linked list of **messages** awaiting processing by a **module** or **driver**.

Read Queue
In a **stream**, the **message queue** in a **module** or **driver** containing **messages** moving **upstream**.

Write Queue
In a **stream**, the **message queue** in a **module** or **driver** containing **messages** moving **downstream**.

Multiplexor

A **driver** that allows **streams** associated with several user processes to be connected to a single **driver** or several **drivers** to be connected to a single user process. **STREAMS** does not provide a general multiplexing **driver**, but provides the facilities for constructing them and for connecting multiplexed **streams** configurations.

Sockets and Address Families

A socket is an endpoint for communication between processes. Each socket has queues for sending and receiving data.

Sockets are typed according to their communications properties. These properties include whether messages sent and received at a socket require the name of the partner, whether communication is reliable, the format used in naming message recipients, etc.

Each instance of the system supports some collection of socket types; consult *socket*(2B) for more information about the types available and their properties.

Each instance of the system supports some number of sets of communications protocols. Each protocol set supports addresses of a certain format. An Address Family is the set of addresses for a specific group of protocols. Each socket has an address chosen from the address family in which the socket was created.

SEE ALSO

intro(3).

NAME

exit, _exit - terminate process

SYNOPSIS

void exit (status)
int status;
void _exit (status)
int status;

DESCRIPTION

exit terminates the calling process with the following consequences:

All file and socket descriptors open in the calling process are closed.

If the parent process of the calling process is executing a *wait*(2), the parent process is notified of the calling process's termination and the low order eight bits (bits 0377) of *status* are made available to it (see *wait*(2)).

If the parent process of the calling process is not executing a *wait*, the calling process is transformed into a zombie process. A zombie process only occupies a slot in the process table. It has no other space allocated either in user or kernel space. The process table slot that it occupies is partially overlaid with time accounting information to be used by *times*(2) (see **<sys/proc.h>**).

The parent process ID of all of the calling processes' existing child processes and zombie processes is set to 1. This means the initialization process inherits each of these processes (see *intro*(2)).

Each attached shared memory segment is detached and the value of *shm_nattach* in the data structure associated with its shared memory identifier is decremented by 1.

For each semaphore for which the calling process has set a *semadj* value (see *semop*(2)), that *semadj* value is added to the *semval* of the specified semaphore.

If the process has a process, text, or data lock, an *unlock* is performed (see *plock*(2)).

An accounting record is written on the accounting file if the system's accounting routine is enabled (see *acct*(2)).

If the process ID, tty group ID, and process group ID of the calling process are equal, the SIGHUP signal is sent to each process that has a process group ID equal to that of the calling process.

If a child or sibling of the calling process was stopped due to a stop signal (see *signal*(2)), the child or sibling will be sent the SIGCONT and SIGHUP signals.

A death of child signal is sent to the parent.

If the calling process is a process group leader (the calling process at some point had called *setpgrp*(2)) and has a controlling terminal with a tty group

ID that does not match the caller's process group ID, the signal SIGHUP will be sent to each process that has a process group ID equal to the tty group ID or the caller's process group ID.

If the calling process is a process group leader that has a controlling terminal, read and write permission will be removed for all processes that have this controlling terminal open.

Any outstanding XIO requests will be canceled and the associated resources will be deallocated.

Any areas of memory locked using *vlock*(2I) will be unlocked.

The C function *exit* may cause cleanup actions before the process exits. The function *_exit* circumvents all cleanup.

SEE ALSO

intro(2), setpgrp(2), signal(2), sigset(2), wait2(2I).
wait(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

There can be no return from an *exit* system call.

WARNINGS

See **WARNINGS** in *signal*(2).

NAME

fcntl - file control

SYNOPSIS

#include <fcntl.h>

int fcntl (fildes, cmd, arg)
int fildes, cmd, arg;

DESCRIPTION

fcntl provides for control over open files. *Fildes* is an open file descriptor obtained from a *creat*(2), *open*(2), *dup*(2), *fcntl*(5), *pipe*(2), *accept*(2B), *socket*(2B), or *socketpair*(2B) system call.

The commands available are as follows:

F_DUPFD Return a new file descriptor as follows:

Lowest-numbered available file descriptor greater than or equal to *arg*.

Same open file (or pipe) as the original file.

Same file pointer as the original file (Both file descriptors share one file pointer.)

Same access mode (read, write or read/write).

Same file status flags. (Both file descriptors share the same file status flags.)

The close-on-exec flag associated with the new file descriptor is set to remain open across *exec*(2) system calls.

F_GETFD Get the close-on-exec flag associated with the file descriptor *fildes*. If the low-order bit is 0, the file will remain open across *exec*(2). Otherwise, the file will be closed when *exec*(2) is executed.

F_SETFD Set the close-on-exec flag associated with *fildes* to the low-order bit of *arg* (0 or 1 as above).

F_GETFL Get *file* status flags.

F_SETFL Set *file* status flags to *arg*. Only certain flags can be set (see *fcntl*(5)).

F_GETLK Get the first lock that blocks the lock description given by the variable of type *struct flock* pointed to by *arg*. The information retrieved overwrites the information passed to *fcntl* in the *flock* structure. If no lock is found that would prevent this lock from being created, the structure is passed back unchanged except for the lock type that will be set to F_UNLCK.

F_SETLK Set or clear a file segment lock according to the variable of type *struct flock* pointed to by *arg* (see *fcntl*(5)). The *cmd* F_SETLK is used to establish read (F_RDLCK) and write (F_WRLCK) locks and remove either type of lock (F_UNLCK). If a read or write lock cannot be set, *fcntl* will return immediately with an error value of -1.

F_SETLKW This *cmd* is the same as F_SETLK except that if a read or write lock is blocked by other locks, the process will sleep until the segment is free to be locked.

F_GETOWN Get the process ID or process group currently receiving SIGIO and SIGURG signals; process groups are returned as negative values.

F_SETOWN Set the process or process group to receive SIGIO and SIGURG signals; process groups are specified by supplying *arg* as negative. Otherwise, *arg* is interpreted as a process ID.

A read lock prevents any process from write locking the protected area. More than one read lock may exist for a given segment of a file at a given time. The file descriptor on which a read lock is placed must have been opened with read access.

A write lock prevents any process from read locking or write locking the protected area. Only one write lock may exist for a given segment of a file at a given time. The file descriptor on which a write lock is placed must have been opened with write access.

The structure *flock* describes the type (*l_type*), starting offset (*l_whence*), relative offset (*l_start*), size (*l_len*), process ID (*l_pid*), and RFS system ID (*l_sysid*) of the file segment to be affected. The process ID and system ID fields are used only with the F_GETLK *cmd* to return the values for a blocking lock. Locks may start and extend beyond the current end of a file, but may not be negative relative to the beginning of the file. A lock may be set to always extend to the end of file by setting *l_len* to zero (0). If such a lock also has *l_whence* and *l_start* set to zero (0), the whole file will be locked. Changing or unlocking a segment from the middle of a larger locked segment leaves two smaller segments for either end. Locking a segment that is already locked by the calling process removes the old lock type and the new lock type takes effect. All locks associated with a file for a given process are removed when a file descriptor for that file is closed by that process or the process holding that file descriptor terminates. Locks are not inherited by a child process in a *fork*(2) system call.

When mandatory file and record locking is active on a file, (see *chmod*(2)), *read*(2) and *write*(2) system calls issued on the file will be affected by the record locks in effect.

fcntl will fail if one or more of the following is true:

[EBADF] *Fildes* is not a valid open file descriptor or the *cmd* is F_GETOWN or F_SETOWN and the descriptor does not reference a socket.

[EINVAL]	*Cmd* is F_DUPFD. *arg* is either negative or greater than or equal to the configured value for the maximum number of open file descriptors allowed each user.
[EINVAL]	*Cmd* is F_GETLK, F_SETLK, or SETLKW and *arg* or the data it points to is not valid.
[EACCES]	*Cmd* is F_SETLK, the type of lock (*l_type*) is a read (F_RDLCK) lock, and the file segment to be locked is write locked by another process or the type is a write (F_WRLCK) lock and the segment of a file to be locked is read or write locked by another process.
[ENOLCK]	*Cmd* is F_SETLK or F_SETLKW, the type of lock is a read or write lock, and no more record locks are available (too many file segments locked) because the system maximum has been exceeded.
[EDEADLK]	*Cmd* is F_SETLKW and the lock is blocked by some lock from another process, and putting the calling-process to sleep, waiting for that lock to become free, would cause a deadlock.
[EFAULT]	*Cmd* is F_SETLK, *arg* points outside the program address space.
[EINTR]	A signal was caught during the *fcntl* system call.
[ENOLINK]	*Fildes* is on a remote machine and the link to that machine is no longer active.
[ESRCH]	*Cmd* is F_SETOWN and the process ID given as an argument is not in use.

SEE ALSO

accept(2B), socket(2B), socketpair(2B), fcntl(5).
close(2), creat(2), dup(2), exec(2), fork(2), open(2), pipe(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, the value returned depends on *cmd* as follows:

F_DUPFD	new file descriptor
F_GETFD	value of flag (only the low-order bit is defined)
F_SETFD	value other than -1
F_GETFL	value of file flags
F_SETFL	value other than -1
F_GETLK	value other than -1
F_SETLK	value other than -1
F_SETLKW	value other than -1
F_GETOWN	value of file descriptor owner
F_SETOWN	value other than -1

Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

WARNINGS

Because in the future the variable *errno* will be set to EAGAIN rather than EACCES when a section of a file is locked by another process, portable application programs should expect and test for either value.

NAME

mount - mount a file system

SYNOPSIS

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

int mount (spec, dir, mflag, fstyp, dataptr, datalen)
char *spec, *dir;
int mflag, fstyp;
caddr_t dataptr;
int datalen;
```

DESCRIPTION

mount requests that a removable file system contained on the block special file *spec* be mounted on the directory *dir*. *Spec* and *dir* are pointers to path names. *Fstyp* is the file system type number. The *sysfs*(2) system call can be used to determine the file system type number. If the MS_FSS flag bit of *mflag* is off, the file system type will default to root file system type. If the bit is on, *fstyp* is used to indicate the file system type. Additionally, if the MS_DATA flag is on in *mflag*, *dataptr* and *datalen* are used to pass mount parameters to the system. If MS_DATA is off or if *dataptr* or *datalen* is zero, no additional data exists. In the normal case of a local mount, *dataptr* should be null. When mounting a Network File System™ (NFS), *dataptr* should point to a structure that describes the NFS mount options.

Upon successful completion, references to the file *dir* will refer to the root directory on the mounted file system.

The low-order bit of *mflag* is used to control write permission on the mounted file system; if 1, writing is forbidden. Otherwise, writing is permitted according to individual file accessibility.

mount may be invoked only by the super-user. It is intended for only the *mount*(1M) utility to use it.

mount will fail if one or more of the following is true:

[EPERM]	The effective user ID is not super-user.
[ENOENT]	Any of the named files does not exist.
[ENOTDIR]	A component of a path prefix is not a directory.
[EREMOTE]	*Spec* is remote and cannot be mounted.
[ENOLINK]	*Path* points to a remote machine and the link to that machine is no longer active.
[EMULTIHOP]	Components of *path* require hopping to multiple remote machines.
[ENOTBLK]	*Spec* is not a block special device.

[ENXIO]	The device associated with *spec* does not exist.
[ENOTDIR]	*Dir* is not a directory.
[EFAULT]	*Spec* or *dir* points outside the allocated address space of the process.
[EBUSY]	*Dir* is currently mounted on, is someone's current working directory, or is otherwise busy.
[EBUSY]	The device associated with *spec* is currently mounted.
[EBUSY]	No more mount table entries exist.
[EROFS]	*Spec* is write protected and *mflag* requests write permission.
[ENOSPC]	The file system state in the super-block is not FsOKAY and *mflag* requests write permission.
[EINVAL]	The super-block has a bad magic number, the *fstyp* is invalid, or *mflag* is invalid.

SEE ALSO

sysfs(2), umount(2), fs(4) in the *UNIX System V Programmer's Reference Manual*.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

read - read from file

SYNOPSIS

```
int read (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

DESCRIPTION

Fildes is a file descriptor obtained from a *creat*(2), *open*(2), *dup*(2), *fcntl*(2), *pipe*(2), *accept*(2B), *socket*(2B), or *accept*(2B) system call.

read attempts to read *nbyte* bytes from the file associated with *fildes* into the buffer pointed to by *buf*.

On devices capable of seeking, the *read* starts at a position in the file given by the file pointer associated with *fildes*. Upon return from *read*, the file pointer is incremented by the number of bytes actually read.

Devices incapable of seeking always read from the current position. The value of a file pointer associated with such a file is undefined.

Upon successful completion, *read* returns the number of bytes actually read and placed in the buffer; this number may be less than *nbyte* if the file is associated with a communication line (see *ioctl*(2) and *termio*(7S)), or if the number of bytes remaining in the file is less than *nbyte* bytes. A value of 0 is returned when an end-of-file is reached.

A *read* from a STREAMS (see *intro*(2)) file can operate in three different modes: "byte-stream" mode, "message-nondiscard" mode, and "message-discard" mode. The default is byte-stream mode. This can be changed using the I_SRDOPT *ioctl*(2) request (see *streamio*(7)), and can be tested with the I_GRDOPT *ioctl*(2). In byte-stream mode, *read* will retrieve data from the stream until it retrieves *nbyte* bytes or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, *read* retrieves data until it has read *nbyte* bytes or until it reaches a message boundary. If the *read* does not retrieve all the data in a message, the remaining data is replaced on the stream and can be retrieved by the next *read* or *getmsg*(2) call. Message-discard mode also retrieves data until it has retrieved *nbyte* bytes, or it reaches a message boundary. However, unread data remaining in a message after the *read* returns are discarded, and are not available for a subsequent *read* or *getmsg*(2).

When attempting to read from a regular file with mandatory file/record locking set (see *chmod*(2)), and a blocking (i.e., owned by another process) write lock is on the segment of the file to be read, one of the following will occur:

If O_NDELAY is set, the read will return a -1 and set errno to EAGAIN.

If O_NDELAY is clear, the read will sleep until the blocking record lock is removed.

When attempting to read from an empty pipe (or FIFO) one of the following will occur:

If O_NDELAY is set, the read will return a 0.

If O_NDELAY is clear, the read will block until data is written to the file or the file is no longer open for writing.

When attempting to read a file associated with a tty that has no data currently available one of the following will occur:

If O_NDELAY is set, the read will return a 0.

If O_NDELAY is clear, the read will block until data becomes available.

When attempting to read a file associated with a stream that has no data currently available one of the following will occur:

If O_NDELAY is set, the read will return a -1 and set errno to EAGAIN.

If O_NDELAY is clear, the read will block until data becomes available.

When reading from a STREAMS file, zero-byte message handling is determined by the current read mode setting. In byte-stream mode, *read* accepts data until it reads *nbyte* bytes, until there is no more data to read, or until a zero-byte message block is encountered. *read* then returns the number of bytes read and places the zero-byte message back on the stream to be retrieved by the next *read* or *getmsg*(2). In the two other modes, a zero-byte message returns a value of 0 and the message is removed from the stream. When a zero-byte message is read as the first message on a stream, a value of 0 is returned regardless of the read mode.

A *read* from a STREAMS file can only process data messages. It cannot process any type of protocol message and will fail if a protocol message is encountered at the stream head.

read will fail if one or more of the following is true:

[EAGAIN]	Mandatory file/record locking was set, O_NDELAY was set, and there was a blocking record lock.
[EAGAIN]	Total amount of system memory available when reading via raw I/O is temporarily insufficient.
[EAGAIN]	No message waiting to be read on a stream and O_NDELAY is flag set.
[EBADF]	*Fildes* is not a valid file descriptor open for reading.

[EBADMSG]	Message waiting to be read on a stream is not a data message.
[EDEADLK]	The read was going to go to sleep and caused a deadlock situation to occur.
[EFAULT]	*Buf* points outside the allocated address space.
[EINTR]	A signal was caught during the *read* system call.
[EINVAL]	Attempted to read from a stream linked to a multiplexor.
[ENOLCK]	The system record lock table was full, so the read could not go to sleep until the blocking record lock was removed.
[ENOLINK]	*Fildes* is on a remote machine and the link to that machine is no longer active.
[EWOULDBLOCK]	The descriptor references a socket marked as nonblocking and the requested operation would block.
[ECONNRESET]	The descriptor references a socket where the connection is broken and there is no more data to read.

A *read* from a STREAMS file also fails if an error message is received at the stream head. In this case, *errno* is set to the value returned in the error message. If a hangup occurs on the stream being read, *read* will continue to operate normally until the stream head read queue is empty. Thereafter, it will return 0.

SEE ALSO

fcntl(2), intro(2), accept(2B), socket(2B), socketpair(2B).
termio(7S) in the *CLIX System Administrator's Reference Manual.*
creat(2), dup(2), ioctl(2), open(2), pipe(2), getmsg(2) in the *UNIX System V Programmer's Reference Manual.*
streamio(7) in the *UNIX System V System Administrator's Reference Manual.*
"Introductory Socket Tutorial" in the *CLIX System Guide.*

DIAGNOSTICS

Upon successful completion, a non-negative integer is returned indicating the number of bytes actually read. Otherwise, a -1 is returned and *errno* is set to indicate the error.

NAME

setpgrp - set process group ID

SYNOPSIS

int setpgrp ()

DESCRIPTION

setpgrp sets the calling process's group ID to its process ID and returns the new process group ID. Once a process calls *setpgrp*, it irrevocably becomes a process group leader.

SEE ALSO

signal(2), intro(2).
exec(2), fork(2), getpid(2), kill(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

setpgrp returns the value of the new process group ID.

NAME

signal - specify what to do on receipt of a signal

SYNOPSIS

```
#include <signal.h>

void (*signal (sig, func)) ()
int sig;
void (*func) ();
```

DESCRIPTION

signal allows the calling process to choose one of three ways to handle the receipt of a specific signal. *Sig* specifies the signal and *func* specifies the choice.

Sig can be assigned any one of the following except SIGKILL or SIGSTOP:

SIGHUP	01	hangup
SIGINT	02	interrupt
SIGQUIT	03[1]	quit
SIGILL	04[1]	illegal instruction (not reset when caught)
SIGTRAP	05[1]	trace trap (not reset when caught)
SIGEMT	07[1]	EMT instruction
SIGFPE	08[1]	floating point exception
SIGKILL	09	kill (cannot be caught or ignored)
SIGBUS	10[1]	bus error
SIGSEGV	11[1]	segmentation violation
SIGSYS	12[1]	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGUSR1	16	user-defined signal 1
SIGUSR2	17	user-defined signal 2
SIGCLD	18[2]	death of a child or child has stopped
SIGPWR	19[2]	power failure
SIGURG	20[5]	urgent condition present on socket
SIGIO	21[5]	I/O is possible on a socket (see *fcntl*(2))
SIGPOLL	22[3]	selectable event pending
SIGSTOP	23	stop (cannot be caught or ignored)
SIGTSTP	24[4]	stop signal generated from keyboard
SIGTTIN	25[4]	background read attempted from control terminal
SIGTTOU	26[4]	background write attempted from control terminal
SIGCONT	27[4]	continue if stopped (cannot be ignored)

Func is assigned one of three values: SIG_DFL, SIG_IGN, or a *function address*. SIG_DFL and SIG_IGN are defined in the include file **signal.h.** Each is a macro that expands to a constant expression of type pointer to function returning *void* and has a unique value that matches no declarable function.

The actions prescribed by the values of *func* are as follows:

SIG_DFL - terminate process on receipt of a signal

When the signal *sig* is received, the receiving process is to be terminated with all of the consequences outlined in *exit*(2). See **Note** [1] below.

SIG_IGN - ignore signal

The signal *sig* is ignored.

Note: the signals SIGKILL and SIGSTOP cannot be ignored.

function address - catch signal

When the signal *sig* is received, the receiving process will execute the signal-catching function pointed to by *func*. The signal number *sig* is passed as the only argument to the signal-catching function. Additional arguments are passed to the signal-catching function for hardware-generated signals. Before entering the signal-catching function, the value of *func* for the caught signal is set to SIG_DFL unless the signal is SIGILL, SIGTRAP, or SIGPWR.

On return from the signal-catching function, the receiving process resumes execution where it was interrupted.

When a signal to be caught occurs during a *read*(2), a *write*(2), *open*(2), or *ioctl*(2) system call on a slow device (like a terminal, but not a file) during a *pause*(2) system call, during a *wait*(2) system call that does not return immediately due to the existence of a previously stopped or zombie process, or any other call that may sleep in the kernel, the signal catching function is executed and then the interrupted system call may return a -1 to the calling process with *errno* set to EINTR.

signal does not catch an invalid function argument, *func*, and results are undefined when an attempt is made to execute the function at the bad address.

Note: The signals SIGKILL and SIGSTOP cannot be caught.

A call to *signal* cancels a pending signal *sig* except for a pending SIGKILL or SIGSTOP signal.

signal fails if *sig* is an illegal signal number (including SIGKILL and SIGSTOP).

Notes

[1] If SIG_DFL is assigned for these signals, in addition to the process being terminated, a "core image" is constructed in the current working directory of the process if the following conditions are met:

The effective user ID and the real user ID of the receiving process are equal.

An ordinary file named **core** exists and is writable or can be created. If the file must be created, it will have the following properties:

A mode of 0666 modified by the file creation mask (see *umask*(2)).

A file owner ID that is the same as the effective user ID of the receiving process.

A file group ID that is the same as the effective group ID of the receiving process.

[2] For the signals SIGCLD and SIGPWR, *func* is assigned one of three values: SIG_DFL, SIG_IGN, or a *function address*. The actions prescribed by these values are:

SIG_DFL - ignore signal
The signal is to be ignored.

SIG_IGN - ignore signal
The signal will be ignored. Also, if *sig* is SIGCLD, the calling process's child processes does not create zombie processes when they terminate (see *exit*(2)).

function address - catch signal
If the signal is SIGPWR, the action to be taken is the same as that described above for *func* equal to *function address*. The same is true if the signal is SIGCLD with one exception: while the process is executing the signal-catching function, any received SIGCLD signals are ignored. (This is the default action.)

In addition, SIGCLD affects the *wait*(2) and *exit*(2) system calls as follows:

wait(2) If the *func* value of SIGCLD is set to SIG_IGN and a *wait* is executed, the *wait* blocks until all of the calling process's child processes terminate; it then returns a value of -1 with *errno* set to ECHILD.

exit(2) If the *func* value of SIGCLD is set to SIG_IGN in the exiting process's parent process, the exiting process does not create a zombie process.

When processing a pipeline, the shell makes the last process in the pipeline the parent of the proceeding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should not set SIGCLD to be caught.

[3] SIGPOLL is issued when a file descriptor corresponding to a STREAMS (see *intro*(2)) file has a "selectable" event pending. A process must specifically request that this signal be sent using the I_SETSIG *ioctl*(2) call. Otherwise, the process will never receive SIGPOLL.

[4] If SIG_DFL is assigned for the SIGTSTP, SIGTTIN or SIGTTOU signals, the process enters the stopped state until a SIGCONT or SIGKILL signal is received. If SIG_DFL is assigned for the SIGCONT signal and the process has entered the stopped state by receiving a stop signal (SIGSTOP, SIGTSTP, SIGTTIN or SIGTTOU), the process returns to its prior state.

[5] The default value for these signals is SIG_IGN.

SEE ALSO

intro(2), sigset(2), sigcld(2I), wait2(2I).
kill(2), pause(2), ptrace(2), setjmp(3C), wait(2) in the *UNIX System V Programmer's Reference Manual.*
kill(1) in the *UNIX System V User's Reference Manual.*

DIAGNOSTICS

Upon successful completion, *signal* returns the previous value of *func* for the specified signal *sig*. Otherwise, a value of SIG_ERR is returned and *errno* is set to indicate the error. SIG_ERR is defined in the include file **<signal.h>**.

NAME

sigset, sighold, sigrelse, sigignore, sigpause - signal management

SYNOPSIS

```
#include <signal.h>

void (*sigset (sig, func)) ()
int sig;
void (*func) ();

int sighold (sig)
int sig;

int sigrelse (sig)
int sig;

int sigignore (sig)
int sig;

int sigpause (sig)
int sig;
```

DESCRIPTION

These functions provide signal management for application processes. *sigset* specifies the system signal action to be taken when signal *sig* is received. This action is either calling a process signal-catching handler *func* or performing a system-defined action.

Sig can be assigned any one of the following values except SIGKILL or SIGSTOP. Machine- or implementation-dependent signals are not included (see **NOTES** below). Each value of *sig* is a macro, defined in **<signal.h>**, that expands to an integer constant expression.

SIGHUP	hangup
SIGINT	interrupt
SIGQUIT*	quit
SIGILL*	illegal instruction (not held when caught)
SIGTRAP*	trace trap (not held when caught)
SIGABRT*	abort
SIGFPE*	floating point exception
SIGKILL	kill (cannot be caught or ignored)
SIGSYS*	bad argument to system call
SIGPIPE	write on a pipe with no one to read it
SIGALRM	alarm clock
SIGTERM	software termination signal
SIGUSR1	user-defined signal 1
SIGUSR2	user-defined signal 2
SIGCLD	death of a child or child has stopped (see **WARNINGS** below)
SIGPWR	power fail (see **WARNINGS** below)
SIGURG	urgent condition present on socket (see **NOTES** below)
SIGIO	I/O is possible on a socket (see **NOTES** below)
SIGPOLL	selectable event is pending (see **NOTES** below)

SIGSTOP	stop (cannot be caught or ignored)
SIGTSTP	stop signal generated from keyboard
SIGTTIN	background read attempted from control terminal
SIGTTOU	background write attempted from control terminal
SIGCONT	continue if stopped (cannot be ignored)

SIG_DFL (below) explains asterisks (*) in the above list.

The following values for the system-defined actions of *func* are also defined in **<signal.h>**. Each is a macro that expands to a constant expression of type pointer to function returning *void* and has a unique value that matches no declarable function.

SIG_DFL - default system action

If SIG_DFL is assigned for the SIGTSTP, SIGTTIN or SIGTTOU signals, the process will enter the stopped state until a SIGCONT or SIGKILL signal is received. If SIG_DFL is assigned for the SIGCONT signal and the process has entered the stopped state by receiving a stop signal (SIGSTOP, SIGTSTP, SIGTTIN or SIGTTOU), the process returns to it's prior state.

Otherwise, upon receipt of the signal *sig*, the receiving process is to be terminated with all of the consequences outlined in *exit*(2). In addition, a "core image" will be made in the current working directory of the receiving process if an asterisk appears with *sig* in the above list *and* the following conditions are met:

The effective user ID and the real user ID of the receiving process are equal.

An ordinary file named core exists and is writable or can be created. If the file must be created, it will have the following properties:

A mode of 0666 modified by the file creation mask (see *umask*(2)).

A file owner ID that is the same as the effective user ID of the receiving process.

A file group ID that is the same as the effective group ID of the receiving process.

SIG_IGN - ignore signal

Any pending signal *sig* is discarded and the system signal action is set to ignore future occurrences of this signal type.

SIG_HOLD - hold signal

The signal *sig* is to be held when it is receipt. Any pending signal of this type remains held. Only one signal of each type is held.

Otherwise, *func* must be a pointer to a function, the signal-catching handler, that is to be called when signal *sig* occurs. In this case, *sigset* specifies that the process will call this function when it receives signal *sig*. Any pending signal of this type is released. This handler address is retained across calls to the other signal management functions listed here.

When a signal occurs, the signal number *sig* will be passed as the only argument to the signal-catching handler. Before calling the signal-catching handler, the system signal action will be set to SIG_HOLD. During normal return from the signal-catching handler, the system signal action is restored to *func* and any held signal of this type released. If a nonlocal goto (*longjmp*) is taken, *sigrelse* must be called to restore the system signal action and release any held signal of this type.

In general, upon return from the signal-catching handler, the receiving process resumes execution where it was interrupted. However, when a signal is caught during a *read*(2), *write*(2), *open*(2), or *ioctl*(2) system call during a *sigpause* system call, or during a *wait*(2) system call that does not return immediately due to a previously stopped or zombie process, or any other call that may sleep in the kernel, the signal-catching handler will be executed and then the interrupted system call may return a -1 to the calling process with *errno* set to EINTR.

sighold and *sigrelse* are used to establish critical regions of code. *sighold* is analogous to raising the priority level and deferring or holding a signal until the priority is lowered by *sigrelse*. *sigrelse* restores the system signal action to that specified previously by *sigset*.

sigignore sets the action for signal *sig* to SIG_IGN (see above).

sigpause suspends the calling process until it receives a signal, the same as *pause*(2). However, if the signal *sig* had been received and held, it is released and the system signal action taken. This system call is useful for testing variables changed on the occurrence of a signal. The correct usage is to use *sighold* to block the signal first, and then test the variables. If they have not changed, call *sigpause* to wait for the signal.

sigset will fail if one or more of the following is true:

[EINVAL]	*Sig* is an illegal signal number (including SIGKILL or SIGSTOP) or the default handling of *sig* cannot be changed.
[EINTR]	A signal was caught during the system call *sigpause*.

SEE ALSO

signal(2), sigcld(2I), wait2(2I).
kill(2), pause(2), setjmp(3C), wait(2) in the *UNIX System V Programmer's Reference Manual*.

DIAGNOSTICS

Upon successful completion, *sigset* returns the previous value of the system signal action for the specified signal *sig*. Otherwise, a value of SIG_ERR is returned and *errno* is set to indicate the error. SIG_ERR is defined in **<signal.h>**.

For the other functions, upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NOTES

SIGPOLL is issued when a file descriptor corresponding to a STREAMS (see *intro*(2)) file has a "selectable" event pending. A process must specifically request that this signal be sent using the I_SETSIG *ioctl*(2) call (see *streamio*(7)). Otherwise, the process will never receive SIGPOLL .

For portability, applications should use only the symbolic names of signals rather than their values and use only the set of signals defined here. The action for the signals SIGKILL and SIGSTOP cannot be changed from the default system action.

Specific implementations may have other implementation-defined signals. Also, additional implementation-defined arguments may be passed to the signal-catching handler for hardware-generated signals. For certain hardware-generated signals, it may not be possible to resume execution at the point of interruption.

SIG_IGN is the default value for the SIGIO and SIGURG signals.

The signal type SIGSEGV is reserved for the condition that occurs on an invalid access to a data object. If an implementation can detect this condition, this signal type should be used.

The other signal management functions, *signal*(2) and *pause*(2), should not be used with these routines for a particular signal type.

WARNINGS

Two signals that behave differently than the signals described above exist in this release of the system:

SIGCLD	death of a child (reset when caught)
SIGPWR	power failure (not reset when caught)

For these signals, *func* is assigned one of three values: SIG_DFL, SIG_IGN, or a *function address*. The actions prescribed by these values are as follows:

SIG_DFL - ignore signal

The signal is to be ignored.

SIG_IGN - ignore signal

The signal is to be ignored. Also, if *sig* is SIGCLD, the calling process's child processes do not create zombie processes when they terminate (see *exit*(2)).

function address - catch signal

If the signal is SIGPWR, the action to be taken is the same as that described above for *func* equal to *function address*. The same is true if the signal is SIGCLD with one exception: while the process is executing the signal-catching function, any received SIGCLD signals will be ignored. (This is the default action.)

The SIGCLD also affects the two system calls *wait*(2) and *exit*(2) in the following ways:

wait(2) If the *func* value of SIGCLD is set to SIG_IGN and a *wait* is executed, the *wait* will block until all of the calling process's child processes terminate; it then returns a value of -1 with *errno* set to ECHILD.

exit(2) If in the exiting process's parent process the *func* value of SIGCLD is set to SIG_IGN , the exiting process will not create a zombie process.

When processing a pipeline, the shell makes the last process in the pipeline the parent of the proceeding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should not set SIGCLD to be caught.

NAME

write - write to a file

SYNOPSIS

```
int write (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

DESCRIPTION

Fildes is a file descriptor obtained from a *creat*(2), *open*(2), *dup*(2), *fcntl*(2), *pipe*(2), *accept*(2B), *socket*(2B), or *socketpair*(2B) system call.

write attempts to write *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the *fildes*.

On devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from *write*, the file pointer is incremented by the number of bytes written.

On devices incapable of seeking, writing always begins at the current position. The value of a file pointer associated with such a device is undefined.

If the O_APPEND file status flag is set, the file pointer will be set to the end of the file before each write.

For regular files, if the O_SYNC file status flag is set, the write will not return until both the file data and file status have been physically updated. This function is for special applications that require extra reliability at the cost of performance. For block special files, if O_SYNC is set, the write will not return until the data has been physically updated.

A write to a regular file will be blocked if mandatory file/record locking is set (see *chmod*(2)), and a record lock is owned by another process on the file segment to be written. If O_NDELAY is not set, the write will sleep until the blocking record lock is removed.

For STREAMS (see *intro*(2)) files, the operation of *write* is determined by the values of the minimum and maximum *nbyte* range ("packet size") accepted by the *stream*. These values are contained in the topmost stream module. Unless the user pushes (see I_PUSH in *streamio*(7)) the topmost module, these values cannot be set or tested from user level. If *nbyte* falls within the packet size range, *nbyte* bytes will be written. If *nbyte* does not fall within the range and the minimum packet size value is zero, *write* breaks the buffer into maximum packet size segments before sending the data downstream. (The last segment may contain less than the maximum packet size.) If *nbyte* does not fall within the range and the minimum value is nonzero, *write* fails with *errno* set to ERANGE. Writing a zero-length buffer (*nbyte* is zero) sends zero bytes with zero returned.

For STREAMS files, if O_NDELAY is not set and the stream cannot accept data because the stream write queue is full due to internal flow control conditions, *write* will block until data can be accepted. O_NDELAY will prevent a process from blocking due to flow control conditions. If O_NDELAY is set and the stream cannot accept data, *write* fails. If O_NDELAY is set and part of the buffer has been written when a condition in which the stream cannot accept additional data occurs, *write* terminates and returns the number of bytes written.

write will fail and the file pointer remains unchanged if one or more of the following is true:

[EAGAIN]	Mandatory file/record locking was set, O_NDELAY was set, and there was a blocking record lock.
[EAGAIN]	Total amount of system memory available when reading via raw I/O is temporarily insufficient.
[EAGAIN]	An attempt to write to a stream that cannot accept data with the O_NDELAY flag set was made.
[EBADF]	*Fildes* is not a valid file descriptor open for writing.
[EDEADLK]	The write was going to go to sleep and cause a deadlock situation to occur.
[EFAULT]	*Buf* points outside the process allocated address space.
[EFBIG]	An attempt was made to write a file that exceeds the process's file size limit or the maximum file size (see *ulimit*(2)).
[EINTR]	A signal was caught during the *write* system call.
[EINVAL]	Attempt to write to a stream linked below a multiplexor.
[ENOLCK]	The system record lock table was full, so the write could not go to sleep until the blocking record lock was removed.
[ENOLINK]	*Fildes* is on a remote machine and the link to that machine is no longer active.
[ENOSPC]	During a *write* to an ordinary file, no free space remains on the device.
[ENXIO]	A hangup occurred on the stream being written to.
[EPIPE] [SIGPIPE]	An attempt is made to write to a pipe that is not open for reading by any process.
[ERANGE]	An attempt is made to write to a stream with *nbyte* outside the specified minimum and maximum write range, and the minimum value is nonzero.
[EWOULDBLOCK]	The descriptor references a socket marked nonblocking and the requested operation would block.

[EPIPE]	The descriptor references a socket that was disconnected or shut down for writing.
[ENOTCONN]	The descriptor references a socket attempting to send data on a stream that is not connected.
[EDESTADDRREQ]	The descriptor references a socket attempting to send data on a datagram without a destination address.

If a *write* requests that more bytes be written than space is available for (such as the *ulimit*(1) or the physical end of a medium), only as many bytes as space is available for will be written. For example, suppose space is available for 20 bytes more in a file before reaching a limit. A write of 512-bytes will return 20. The next write of a nonzero number of bytes will give a failure return (except as noted below).

If the file being written is a pipe (or FIFO) and the O_NDELAY flag of the file flag word is set, *write* to a full pipe (or FIFO) will return a count of 0. Otherwise, (O_NDELAY clear), writes to a full pipe (or FIFO) will block until space becomes available.

A write to a STREAMS file can fail if an error message was received at the stream head. In this case, *errno* is set to the value included in the error message.

SEE ALSO

fcntl(2), intro(2), accept(2B), socket(2B), socketpair(2B).
creat(2), dup(2), lseek(2), open(2), pipe(2), ulimit(2) in the *UNIX System V Programmer's Reference Manual.*
"Introductory Socket Tutorial" in the *CLIX System Guide.*

DIAGNOSTICS

Upon successful completion, the number of bytes written is returned. Otherwise, -1 is returned and *errno* is set to indicate the error.

2B

NAME

accept - accept a connection on a socket

SYNOPSIS

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

int accept (s, addr, addrlen)
int s;
struct sockaddr *addr;
int *addrlen;
```

DESCRIPTION

The argument *s* is a socket created with *socket*(2B), bound to an address with *bind*(2B), and listening for connections after a *listen*(2B). *accept* extracts the first connection on the queue of pending connections, creates a new socket with the same properties of *s*, and allocates a new file descriptor for the socket. If no pending connections are present on the queue and the socket is marked as nonblocking, *accept* returns an error as described below. The accepted socket may not be used to accept more connections. The original socket *s* remains open.

The argument *addr* is a result parameter filled in with the address of the connecting entity as known to the communications layer. The exact format of the *addr* parameter is determined by the domain in which the communication is occurring. The *addrlen* is a value-result parameter; it should initially contain the amount of space pointed to by *addr*. On return, it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types. It is currently used with SOCK_STREAM.

It is possible to *select*(2B) a socket for the purposes of doing an *accept* by selecting it for read.

accept will fail if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[EOPNOTSUPP]	The referenced socket is not of type SOCK_STREAM.
[EFAULT]	The *addr* parameter is not in a writable part of the user address space.
[EMFILE]	The per-process descriptor table is full.
[ENFILE]	The system file table is full.
[ENOBUFS]	The resources to support this connection are not available.
[ENXIO]	The maximum number of open devices was exceeded.

[EINVAL]	The *addrlen* parameter is an invalid size or the socket is not listening.
[EISCONN]	The socket is already connected.
[EWOULDBLOCK]	The socket is marked nonblocking and no connections are present to be accepted.
[EINTR]	A signal was caught during the *accept* system call.

SEE ALSO

bind(2B), connect(2B), listen(2B), select(2B), socket(2B).

DIAGNOSTICS

Upon successful completion, the new socket descriptor is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

bind - bind a name to a socket.

SYNOPSIS

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

int bind (s, name, namelen)
int s, namelen;
struct sockaddr *name;
```

DESCRIPTION

bind assigns a name to unnamed socket *s*. When a socket is created with *socket*(2B) it exists in a name space (address family) but has no name assigned. *bind* requests that *name* be assigned to the socket. The exact format of the *name* parameter is determined by the domain in which the communication will occur. The *namelen* parameter contains the amount of space pointed to by *name*.

bind will fail if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[EINVAL]	The *namelen* parameter is not the expected size or the socket is already bound to an address.
[EFAULT]	The *name* parameter is not in a valid part of the user address space.
[EADDRNOTAVAIL]	The specified address is not available from the local machine.
[EADDRINUSE]	The specified address is in use.
[EACCES]	The requested address is protected, and the current user has inadequate permission to access it.

The following errors are specific to binding names in the UNIX domain.

[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	A prefix component of the path name does not exist.
[EIO]	An I/O error occurred while making the directory entry or allocating the i-node.
[EROFS]	The name would reside on a read-only file system.
[EISDIR]	A null path name was specified.

SEE ALSO

connect(2B), listen(2B), socket(2B), getsockname(2B).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NOTES

Binding a name in the UNIX domain creates a socket in the file system that must be deleted by the caller when it is no longer needed (using *unlink*(2)).

NAME

connect - initiate a connection on a socket

SYNOPSIS

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

int connect (s, name, namelen)
int s, namelen;
struct sockaddr *name;
```

DESCRIPTION

The parameter *s* is a socket. If it is of type SOCK_DGRAM, this call specifies the peer the socket is to be associated with. This is the address datagrams will be sent to and the only address datagrams will be received from. If the socket is of type SOCK_STREAM, this call attempts to connect to another socket. The other socket is specified by *name*, which is an address in the communications space of the socket. Each communications space interprets the *name* parameter in its own way. Generally, stream sockets may successfully *connect* only once; datagram sockets may use *connect* multiple times to change their association. Datagram sockets may dissolve the association by connecting to an invalid address, such as a null address.

connect fails if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[EINVAL]	The *namelen* parameter is not the expected size.
[EFAULT]	The *name* parameter specifies an area outside the user address space.
[EADDRNOTAVAIL]	The specified address is not available on this machine.
[EISCONN]	The socket is already connected.
[ECONNREFUSED]	The attempt to connect was forcefully rejected.
[ENETUNREACH]	The network cannot be reached from this host.
[EADDRINUSE]	The address is already in use.

The following errors are specific to connecting names in the UNIX domain. These errors may not apply in future versions of the UNIX IPC domain.

[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	The named socket does not exist.
[EACCES]	Search permission is denied for a component of the path prefix or write access to the named socket is denied.

SEE ALSO

accept(2B), select(2B), socket(2B), getsockname(2B).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

ftruncate - truncate a file to a specified length

SYNOPSIS

#include <sys/types.h>

int ftruncate (fd, length)
int fd;
off_t length;

DESCRIPTION

ftruncate truncates the file referenced by *fd* to at most *length* bytes. If the file was larger than *length* bytes, the extra data is lost. The file must be opened for writing.

ftruncate fails if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[EINVAL]	The file is not opened for writing or the descriptor references a socket, not a file.
[EIO]	An I/O error occurred reading a block to be zeroed.

SEE ALSO

open(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

getdtablesize - get descriptor table size

SYNOPSIS

int getdtablesize ()

DESCRIPTION

Each process has a fixed-size file descriptor table that is guaranteed to have at least 20 slots. *getdtablesize* returns the size of this table.

NAME

gethostid, sethostid - get/set unique identifier of current host

SYNOPSIS

```
int gethostid ()

int sethostid (hostid)
long hostid;
```

DESCRIPTION

gethostid returns the 32-bit identifier for the host machine. No errors are possible.

sethostid establishes a 32-bit identifier for the host machine intended to be unique among all UNIX systems in existence. This is normally a Defense Advanced Research Project Agency (DARPA) Internet address for the local machine.

sethostid fails if the following is true:

[EPERM] The caller is not the super-user.

SEE ALSO

gethostname(2B).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NOTES

Only the super-user can execute this call.

NAME

gethostname, sethostname - get/set name of current host

SYNOPSIS

```
int gethostname (name, namelen)
char *name;
int namelen;

int sethostname (name, namelen)
char *name;
int namelen;
```

DESCRIPTION

gethostname returns the standard host name for the host processor. The parameter *namelen* specifies the size of the *name* array. The returned name is null-terminated unless insufficient space is provided.

sethostname establishes the name of the host machine to be *name*. This call is allowed only to the super-user and is performed at boot time.

These calls fail if one or more of the following is true:

[EFAULT]	The *name* parameter specified an invalid address in the user address space.
[EPERM]	The caller is not the super-user.
[EINVAL]	The *namelen* parameter for *sethostname* is larger than the maximum size of a host name.

SEE ALSO

gethostid(2B).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NOTES

The maximum length for a host name is 64 characters.

NAME

getitimer, setitimer - get/set value of interval timer

SYNOPSIS

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

getitimer (which, value)
int which;
struct itimerval *value;

setitimer (which, value, ovalue)
int which;
struct itimerval *value, *ovalue;
```

DESCRIPTION

The system provides each process with three interval timers, defined in **<sys/time.h>**. The *getitimer* call returns the current value for the timer specified in *which* in the structure at *value*. The *setitimer* call sets a timer to the specified *value* (returning the previous timer value if *ovalue* is nonzero).

A timer value is defined by the *itimerval* structure:

```
struct itimerval {
        struct  timeval it_interval;    /* timer interval */
        struct  timeval it_value;       /* current value */
};
```

A nonzero *it_value* indicates the time until the next timer expiration. A nonzero *it_interval* specifies a value to be used in reloading *it_value* when the timer expires. Setting *it_value* to 0 disables a timer. Setting *it_interval* to 0 disables a timer after its next expiration (assuming *it_value* is nonzero).

Time values smaller than the system clock resolution are rounded up to this resolution (1/60-second interval).

The ITIMER_REAL timer decrements in real time. A SIGALRM signal is delivered when this timer expires.

The ITIMER_VIRTUAL timer decrements in process virtual time. It runs only when the process is executing. A SIGVTALRM signal is delivered when this timer expires.

The ITIMER_PROF timer decrements both in process virtual time and when the system is running on behalf of the process. It is designed for interpreters to use in statistically profiling the execution of interpreted programs. Each time the ITIMER_PROF timer expires, the SIGPROF signal is delivered. Because this signal may interrupt in-progress system calls, programs using this timer must be prepared to restart interrupted system calls.

getitimer and *setitimer* will fail if one or more of the following are true:

[EFAULT]	The *value* parameter specified a bad address.
[EINVAL]	A *value* parameter specified a time too large to be handled.

SEE ALSO

sigset(2), gettimeofday(2B).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NOTES

Three macros for manipulating time values are defined in **<sys/time.h>**. *Timerclear* sets a time value to zero, *timerisset* determines whether a time value is nonzero, and *timercmp* compares two time values (>= and <= do not work with this macro).

NAME

getpagesize - get system page size

SYNOPSIS

int getpagesize ()

DESCRIPTION

getpagesize returns the number of bytes in a system page. Page granularity is the granularity of the memory management calls.

NAME

getpeername - get name of connected peer

SYNOPSIS

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

int getpeername (s, name, namelen)
int s;
struct sockaddr *name;
int *namelen;
```

DESCRIPTION

getpeername returns the name of the peer connected to socket *s*. The *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes). The name is truncated if the buffer provided is too small.

getpeername will fail if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[ENOTCONN]	The socket is not connected.
[EFAULT]	The *name* parameter points to memory not in a valid part of the user address space.

SEE ALSO

accept(2B), bind(2B), socket(2B), getsockname(2B).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

getpgrp2 - get process group

SYNOPSIS

int getpgrp2 (pid)
int pid;

DESCRIPTION

The process group of the specified process is returned by *getpgrp2*. If *pid* is zero, the call applies to the current process.

Process groups are used to distribute signals, and by are used terminals to arbitrate requests for their input. Processes that have the same process group as the terminal are foreground and may read, while others will block with a signal if they attempt to read.

getpgrp2 will fail if the following is true:

[ESRCH] The requested process does not exist.

SEE ALSO

setpgrp2(2B), setpgrp(2).
getpgrp(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, the process group of the specified process is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

getsockname - get socket name

SYNOPSIS

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

int getsockname (s, name, namelen)
int s;
struct sockaddr *name;
int *namelen;
```

DESCRIPTION

getsockname returns the current *name* for the specified socket obtained from *socket*(2B). The *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return, it contains the actual size of the name returned (in bytes).

getsockname will fail if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[EFAULT]	The *name* parameter points to memory not in a writable part of the user address space.

SEE ALSO

bind(2B), socket(2B).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NOTES

Names bound to sockets in the UNIX domain are inaccessible; *getsockname* returns a zero-length name.

NAME

getsockopt, setsockopt - get and set options on sockets

SYNOPSIS

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

int getsockopt (s, level, optname, optval, optlen)
int s, level, optname;
char *optval;
int *optlen;

int setsockopt (s, level, optname, optval, optlen)
int s, level, optname;
char *optval;
int *optlen;
```

DESCRIPTION

getsockopt and *setsockopt* manipulate options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost "socket" level.

When manipulating socket options, the level at which the option resides and the name of the option must be specified. To manipulate options at the "socket" level, *level* is specified as SOL_SOCKET. To manipulate options at any other level, the protocol number of the appropriate protocol controlling the option is supplied.

The parameters *optval* and *optlen* are used to access option values for *setsockopt*. For *getsockopt* they identify a buffer in which the value for the requested option(s) will be returned. For *getsockopt*, *optlen* is a value-result parameter, initially containing the size of the buffer pointed to by *optval*, and modified on return to indicate the actual size of the value returned. If no option value will be supplied or returned, *optval* may be supplied as 0.

Optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file **<sys/socket.h>** contains definitions for "socket"-level options, described below. Options at other protocol levels vary in format and name. Consult the appropriate entries in section (7B).

Most socket-level options take an **int** parameter for *optval*. For *setsockopt*, the parameter should be nonzero to enable a boolean option, or zero to disable the option.

The following options are recognized at the "socket" level. Except as noted, each may be examined with *getsockopt* and set with *setsockopt*.

SO_DEBUG	Toggle recording of debugging information.
SO_REUSEADDR	Toggle local address reuse.
SO_KEEPALIVE	Toggle keeping connections alive.

SO_DONTROUTE	Toggle routing bypass for outgoing messages.
SO_LINGER	Linger on close if data is present.
SO_BROADCAST	Toggle permission to transmit broadcast messages.
SO_OOBINLINE	Toggle reception of out-of-band data in band.
SO_TYPE	Get the type of the socket (*get* only).

SO_DEBUG enables debugging in the underlying protocol modules. SO_REUSEADDR indicates that the rules used in validating addresses supplied in a *bind*(2B) call should allow local addresses to be reused. SO_KEEPALIVE enables the periodic transmission of messages on a connected socket. If the connected party fails to respond to these messages, the connection is considered broken and processes using the socket are notified with a SIGPIPE signal. SO_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

SO_LINGER controls the action taken when unsent messages are queued on the socket and a *close*(2) is performed. If the socket promises reliable delivery of data and SO_LINGER is set, the system will block the process on the *close*(2) attempt until it is able to transmit the data or until it decides it is unable to deliver the information. (A timeout period, termed the "linger interval", is specified in the *setsockopt* call when SO_LINGER is requested.) If SO_LINGER is disabled and a *close*(2) is issued, the system will process the close in a manner that allows the process to continue as quickly as possible.

The option SO_BROADCAST requests permission to send broadcast datagrams on the socket. With protocols that support out-of-band data, the SO_OOBINLINE option requests that out-of-band data be placed in the normal data input queue as received; it can then be accessed with *recv*(2B) or *read*(2) calls without the MSG_OOB flag. Finally, SO_TYPE is an option used only with *getsockopt*. SO_TYPE returns the type of the socket, such as SOCK_STREAM; it is useful for servers that inherit sockets on startup.

getsockopt and *setsockopt* will fail if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[EFAULT]	One of the option parameters is not in a valid part of the user address space.
[EINVAL]	The *optlen* is larger than the maximum option length (*setsockopt*) or the specified option does not exists for the given protocol.
[ENOPROTOOPT]	The specified option is unknown at the level indicated.

SEE ALSO

read(2), socket(2B), getprotoent(3B), bind(2B), recv(2B).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

CAVEATS

SO_SNDBUF, SO_RCVBUF and SO_ERROR are not supported in this implementation. An attempt to *set* or *get* one of these options will result in *errno* being set to ENOPROTOOPT.

SO_DEBUG and SO_DONTROUTE are always off in this implementation. Attempting to turn them on will result in *errno* being set to ENOPROTOOPT. Attempting to turn them off will succeed.

SO_KEEPALIVE and SO_BROADCAST are always on in this implementation. Attempting to turn them off will result in *errno* being set to ENOPROTOOPT. Attempting to turn them on will succeed.

SO_LINGER is always on in this implementation. Attempting to turn linger off will result in *errno* being set to ENOPROTOOPT. The linger interval is always set to -1 meaning block indefinitely until all data is transmitted.

NAME

gettimeofday - get date and time

SYNOPSIS

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

int gettimeofday (tp, tzp)
struct timeval *tp;
struct timezone *tzp;
```

DESCRIPTION

The system's notion of the current Greenwich time and the current time zone is obtained with the *gettimeofday* call. The time is expressed in seconds and microseconds since midnight (0 hour), January 1, 1970. The resolution of the system clock is hardware-dependent, and the time may be updated continuously or in "ticks". If *tzp* is zero, the time zone information will not be returned or set.

The structures pointed to by *tp* and *tzp* are defined in **<sys/time.h>** as follows:

```
struct timeval {
        long    tv_sec;           /* seconds since Jan. 1, 1970 */
        long    tv_usec;          /* and microseconds */
};

struct timezone {
        int tz_minuteswest;       /* of Greenwich */
        int tz_dsttime;           /* type of dst correction to apply */
};
```

The *timezone* structure indicates the local time zone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that daylight savings time applies locally during the appropriate part of the year.

gettimeofday will fail if the following is true:

[EFAULT] *Tp* or *tzp* points to an invalid address.

SEE ALSO

time(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

BUGS

Microsecond granularity is not available. Therefore, the *tv_usec* field of the *timeval* structure will always be zero.

NAME

killpg - send signal to a process group

SYNOPSIS

int killpg (pgrp, sig)
int pgrp, sig;

DESCRIPTION

killpg sends the signal *sig* to the process group *pgrp*. See *signal*(2) for a list of signals.

The sending process and members of the process group must have the same effective user ID, or the sender must be the super-user. As a special case, the continue signal SIGCONT may be sent to any process that is a descendant of the current process.

killpg will fail and no signal is sent if any of the following occurs:

[EINVAL]	*Sig* is not a valid signal number or *sig* is SIGKILL and *pgrp* is 1 (process 1).
[ESRCH]	No process can be found in the process group specified by *pgrp*.
[EPERM]	The sending process is not the super-user and one or more of the target processes has an effective user ID different from that of the sending process.

SEE ALSO

signal(2).
kill(2), getpgrp(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the global variable *errno* is set to indicate the error.

NAME

listen - listen for connections on a socket

SYNOPSIS

int listen (s, backlog)
int s, backlog;

DESCRIPTION

To accept connections, a socket is first created with *socket*(2B). A willingness to accept incoming connections and a queue limit for incoming connections are specified with *listen*. Then the connections are accepted with *accept*(2B). The *listen* call applies only to sockets with type SOCK_STREAM.

The *backlog* parameter defines the maximum length for the queue of pending connections. If a connection request arrives with the queue full, the client may receive an error with an indication of ECONNREFUSED, or, if the underlying protocol supports retransmission, the request may be ignored so that retries may succeed.

listen will fail if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[EOPNOTSUPP]	The referenced socket is not of type SOCK_STREAM.

SEE ALSO

accept(2B), connect(2B), socket(2B).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

CAVEATS

Backlog is set automatically by the system to 1.

NAME

lstat - get file status

SYNOPSIS

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

int lstat (path, buf)
char *path;
struct stat *buf;
```

DESCRIPTION

Path points to the path name of a file. Read, write, or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.

Buf is a pointer to a *stat* structure into which information is placed concerning the file.

lstat obtains information about the named file except when the named file is a symbolic link. In this case, *lstat* returns information about the link.

The contents of the structure pointed to by *buf* include the following members:

```
ushort  st_mode;   /* File mode (see mknod(2)) */
ino_t   st_ino;    /* I-node number */
dev_t   st_dev;    /* ID of device containing */
                   /* a directory entry for this file */
dev_t   st_rdev;   /* ID of device */
                   /* This entry is defined only for */
                   /* character special or block special files */
short   st_nlink;  /* Number of links */
ushort  st_uid;    /* User ID of the file's owner */
ushort  st_gid;    /* Group ID of the file's group */
off_t   st_size;   /* File size in bytes */
time_t  st_atime;  /* Time of last access */
time_t  st_mtime;  /* Time of last data modification */
time_t  st_ctime;  /* Time of last file status change */
                   /* Times measured in seconds since */
                   /* 00:00:00 GMT, Jan. 1, 1970 */
```

st_mode The mode of the file as described in the *mknod*(2) system call.

st_ino This uniquely identifies the file in a given file system. The pair *st_ino* and *st_dev* uniquely identifies regular files.

st_dev This uniquely identifies the file system that contains the file. Its value may be used as input to the *ustat*(2) system call to determine more information about this file system. No other meaning is associated with this value.

st_rdev	This should be used only by administrative commands. It is valid only for block special or character special files and only has meaning on the system where the file was configured.
st_nlink	This should be used only by administrative commands.
st_uid	The user ID of the file's owner.
st_gid	The group ID of the file's group.
st_size	For regular files, this is the address of the end of the file. For pipes or FIFOs, this is the count of the data currently in the file. For block special or character special, this is not defined.
st_atime	Time when file data was last accessed. It is changed by the following system calls: *creat*(2), *mknod*(2), *pipe*(2), *symlink*(2B), *utime*(2), and *read*(2).
st_mtime	Time when data was last modified. It is changed by the following system calls: *creat*(2), *mknod*(2), *pipe*(2), *symlink*(2B), *utime*(2), and *write*(2).
st_ctime	Time when file status was last changed. It is changed by the following system calls: *chmod*(2), *chown*(2), *creat*(2), *link*(2), *mknod*(2), *pipe*(2), *symlink*(2B), *unlink*(2), *utime*(2), and *write*(2).

lstat will fail if one or more of the following is true:

[ENOTDIR]	A component of the *path* prefix is not a directory.
[ENOENT]	The named file does not exist or too many symbolic links were in the *path*.
[EACCES]	Search permission is denied for a component of the *path* prefix.
[EFAULT]	*Buf* or *path* points to an invalid address.
[EINTR]	A signal was caught during the *stat* system call.
[ENOLINK]	*Path* points to a remote machine and the link to that machine is no longer active.
[EMULTIHOP]	Components of *path* require hopping to multiple remote machines.

SEE ALSO

chmod(1), chown(1), read(2), write(2), symlink(2B), stat(5).
creat(2), link(2), mknod(2), pipe(2), time(2), unlink(2), utime(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

readlink - read the value of a symbolic link

SYNOPSIS

```
int readlink (path, buf, bufsize)
char *path; *buf;
int bufsize;
```

DESCRIPTION

Path points to the path name of a file. Read, write, or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable. *Buf* is a pointer to the location where the file name will be placed. *Bufsize* is the size of *buf*.

readlink places the contents of the symbolic link name in the buffer *buf* with size *bufsize*. The contents of the link are not null terminated when returned.

readlink will fail if one or more of the following is true:

[ENOTDIR]	A component of the *path* prefix is not a directory.
[ENOENT]	The named file does not exist or too many symbolic links were in the *path*.
[EACCES]	Search permission is denied for a component of the *path* prefix.
[EFAULT]	*Buf* or *path* points to an invalid address.
[EINTR]	A signal was caught during the *readlink* system call.
[ENOLINK]	*Path* points to a remote machine and the link to that machine is no longer active.
[EMULTIHOP]	Components of *path* require hopping to multiple remote machines.

SEE ALSO

lstat(2B), symlink(2B).
stat(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion the count of characters placed, in the buffer is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

readv - read input from a socket

SYNOPSIS

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

int readv (s, iov, iovcnt)
int s, iovcnt;
struct iovec *iov;
```

DESCRIPTION

readv attempts to read data from the object referenced by the descriptor *s*. The input data is scattered into the *iovcnt* buffers specified by the members of the *iov* array: iov[0], iov[1], ..., iov[*iovcnt* - 1].

The *iovec* structure is defined as

```
struct iovec {
        caddr_t    iov_base;
        int        iov_len;
};
```

Each *iovec* entry specifies the base address and length of an area in memory where data should be placed. *readv* will always fill an area completely before proceeding to the next.

If the returned value is 0, end-of-file has been reached.

readv fails if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[EWOULDBLOCK]	The socket is marked nonblocking and the requested operation would block.
[EINTR]	A signal was caught during the *readv* system call.
[EFAULT]	The address specified is not in a valid part of the users address space.
[ECONNRESET]	The connection has been broken and there is no more data to read.
[EINVAL]	The maximum number of scatter gather locations has been exceeded.

SEE ALSO

fcntl(2), read(2), send(2B), select(2B), getsockopt(2B), socket(2B).

DIAGNOSTICS

Upon successful completion, the number of bytes read is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

CAVEATS

readv is only supported on sockets, not on regular files.

NAME

recv, recvfrom, recvmsg - receive a message from a socket

SYNOPSIS

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

int recv (s, buf, len, flags)
int s, len, flags;
char *buf;

int recvfrom (s, buf, len, flags, from, fromlen)
int s, len, flags;
char *buf;
struct sockaddr *from;
int *fromlen;

int recvmsg (s, msg, flags)
int s, flags;
struct msghdr msg[];
```

DESCRIPTION

recv, *recvfrom*, and *recvmsg* are used to receive messages from a socket.

The *recv* call is normally used only on a connected socket (see *connect*(2B)), while *recvfrom* and *recvmsg* may be used to receive data on a socket whether it is in a connected state or not.

If *from* is nonzero, the source address of the message is filled in. *Fromlen* is a value-result parameter initialized to the size of the buffer associated with *from* and modified on return to indicate the actual size of the address stored there. The length of the message is returned by the call. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see *socket*(2B)).

If no messages are available at the socket, the receive call waits for a message to arrive unless the socket is marked nonblocking (see *fcntl*(2)). The *select*(2B) call may be used to determine when more data is available to read.

The *flags* argument to a *recv* call may have the following value:

```
#define MSG_OOB        0x1     /* process out-of-band data */
```

The *recvmsg* call uses a *msghdr* structure to minimize the number of directly supplied parameters. This structure has the following form, defined in **<sys/socket.h>**:

```
struct msghdr {
          caddr_t        msg_name;        /* optional address */
          int            msg_namelen;     /* size of address */
          struct iovec   *msg_iov;        /* scatter/gather array */
          int            msg_iovlen;      /* # elements in msg_iov */
          caddr_t        msg_accrights;   /* access rights sent/received */
```

```
            int         msg_accrightslen;
    };
```

msg_name and *msg_namelen* specify the destination address if the socket is unconnected; *msg_namelen* may be given as a null pointer if no names are desired or required. The *msg_iov* and *msg_iovlen* describe the scatter gather locations, as described in *readv*(2B). A buffer to receive any access rights sent with the message is specified in *msg_accrights*, which has length *msg_accrightslen*. Access rights are currently limited to file descriptors, which each occupy the size of an integer.

Each of these calls fails if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[EWOULDBLOCK]	The socket is marked nonblocking and the requested operation would block.
[EINTR]	A signal was caught during the *recv*, *recvfrom*, or *recvmsg* system call.
[EFAULT]	The data was specified to be received into a nonexistent or protected part of the user address space.
[ECONNRESET]	The connection has been broken and there is no more data to read.
[EINVAL]	The maximum number of scatter gather locations has been exceeded.

SEE ALSO

fcntl(2), readv(2B), send(2B), select(2B), getsockopt(2B), socket(2B).

DIAGNOSTICS

Upon successful completion, the number of bytes received is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

rename - change the name of a file

SYNOPSIS

```
int rename (from, to)
char *from, *to;
```

DESCRIPTION

rename causes the link named *from* to be renamed *to*. If *to* exists, it is removed. Both *from* and *to* must be of the same type (that is, both directories or both nondirectories) and must reside on the same file system.

If the final component of *from* is a symbolic link, the symbolic link, not the file or directory it points to, is renamed.

rename will fail and neither argument file will be affected if any of the following is true:

[ENOTDIR]	A component of either path prefix is not a directory.
[ENOENT]	A component of either path prefix does not exist or *to* points to a null path name.
[EACCES]	A component of either path prefix denies search permission.
[EPERM]	The file named by *from* is a directory and the effective user ID is not super-user.
[EXDEV]	The link named by *to* and the file named by *from* are on different logical devices (file systems).
[EROFS]	The requested link requires writing in a directory on a read-only file system.
[EFAULT]	*From* or *to* point outside the allocated address space of the process.
[EMLINK]	The maximum number of links to a file would be exceeded.
[EINTR]	A signal was caught during the *rename* system call.

SEE ALSO

open(2), link(2), unlink(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

CAVEATS

rename does not guarantee that an instance of *to* will always exist. (For example, if the system should crash in the middle of the operation.)

NAME

select - synchronous I/O multiplexing

SYNOPSIS

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

int select (nfds, readfds, writefds, exceptfds, timeout)
int nfds;
fd_set *readfds, *writefds, *exceptfds;
struct timeval *timeout;

FD_SET (fd, fdset)
FD_CLR (fd, fdset)
FD_ISSET (fd, fdset)
FD_ZERO (fdset)
int fd;
fd_set *fdset;
```

DESCRIPTION

select examines the I/O descriptor sets whose addresses are passed in *readfds*, *writefds*, and *exceptfds* to see if some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. The first *nfds* descriptors are checked in each set. (The descriptors from 0 through *nfds*-1 in the descriptor sets are examined.) On return, *select* replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. The total number of ready descriptors in all the sets is returned.

The descriptor sets are stored as bit fields in arrays of integers. The following macros are provided for manipulating such descriptor sets. *FD_ZERO* initializes the descriptor set *fdset* to the null set. *FD_SET* includes a particular descriptor *fd* in *fdset*. *FD_CLR* removes *fd* from *fdset*. *FD_ISSET* is nonzero if *fd* is a member of *fdset*, zero otherwise. The behavior of these macros is undefined if a descriptor value is less than zero or greater than or equal to FD_SETSIZE, which is normally at least equal to the maximum number of descriptors supported by the system.

If *timeout* is a nonzero pointer, it specifies a maximum interval to wait for the selection to complete. If *timeout* is a zero pointer, the *select* blocks indefinitely. To affect a poll, the *timeout* argument should be nonzero, pointing to a zero-valued timeval structure.

Readfds, *writefds*, or *exceptfds* may be given as zero pointers if no descriptors are of interest.

select fails if one or more of the following is true:

[EBADF]	The descriptor is invalid.

[EINVAL]	The *nfds* parameter is not a valid number.
[EFAULT]	The *readfds*, *writefds*, *exceptfds*, or *timeout* parameters point to a nonwritable area of the user address space.
[EINTR]	A signal was delivered before the time limit expired and before any of the selected events occurred.

SEE ALSO

accept(2B), connect(2B), readv(2B), writev(2B), recv(2B), send(2B).

DIAGNOSTICS

Upon successful completion, the number of ready descriptors is returned. If the time limit expires, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

CAVEATS

select is only supported for sockets and pseudo terminals.

NAME

send, sendto, sendmsg - send a message from a socket

SYNOPSIS

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

int send (s, msg, len, flags)
int s, len, flags;
char *msg;

int sendto (s, msg, len, flags, to, tolen)
int s, len, flags, tolen;
char *msg;
struct sockaddr *to;

int sendmsg (s, msg, flags)
int s, flags;
struct msghdr msg[];
```

DESCRIPTION

send, *sendto*, and *sendmsg* are used to transmit a message to another socket. *send* may be used only when the socket is in a connected state, while *sendto* and *sendmsg* may be used any time.

The address of the target is given by *to*, with *tolen* specifying the size. The length of the message is given by *len*.

No indication of failure to deliver is implicit in a *send*. Return values of -1 indicate some locally detected errors.

If no message space is available at the socket to hold the message to be transmitted, *send* normally blocks unless the socket has been placed in non-blocking I/O mode. The *select*(2B) call may be used to determine when it is possible to send more data.

The *flags* parameter may be set to the following:

```
        #define MSG_OOB        0x1     /* process out-of-band data */
```

The flag MSG_OOB is used to send "out-of-band" data on sockets that support this notion (such as SOCK_STREAM); the underlying protocol must also support "out-of-band" data.

See *recv*(2B) for a description of the *msghdr* structure.

Each of these calls will fail if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[EWOULDBLOCK]	The socket is marked nonblocking and the requested operation would block.
[EFAULT]	An invalid user space address was specified.

[EPIPE]	The connection has been broken and no more data can be sent.
[EINVAL]	The maximum number of scatter gather locations has been exceeded.
[ENOTCONN]	Attempting to send data on a stream socket that is not connected.
[EDESTADDRREQ]	Attempting to send data on a datagram socket without a destination address.
[EINTR]	A signal was caught during the *send*, *sendto*, or *sendmsg* system calls.
[EISCONN]	Attempting to do a *sendto* on a connected socket.

SEE ALSO

fcntl(2), recv(2B), writev(2B), select(2B), getsockopt(2B), socket(2B).

DIAGNOSTICS

Upon successful completion, the number of bytes sent is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

setpgrp2 - set process group

SYNOPSIS

int setpgrp2 (pid, pgrp)
int pid, pgrp;

DESCRIPTION

setpgrp2 sets the process group of the specified process *pid* to the specified *pgrp*. If *pid* is zero, the call applies to the current process.

If the invoker is not the super-user, the affected process must have the same effective user ID as the invoker or be a descendant of the invoking process.

setpgrp2 will fail and the process group will not be altered if one of the following occurs:

[ESRCH]	The requested process does not exist.
[EPERM]	The effective user ID of the requested process is different from that of the caller and the process is not a descendent of the calling process.

SEE ALSO

getpgrp2(2B), setpgrp(2).
getpgrp(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

shutdown - shut down part of a full-duplex connection

SYNOPSIS

int shutdown (s, how)
int s, how;

DESCRIPTION

The *shutdown* call causes all or part of a full-duplex connection on the socket associated with descriptor *s* to be shut down. If *how* is 0, further receives will be disallowed. If *how* is 1, further sends will be disallowed. If *how* is 2, further sends and receives will be disallowed.

shutdown will fail if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[ENOTCONN]	The specified socket is not connected.
[EINVAL]	The value for *how* is not 0, 1, or 2.
[EINTR]	A signal was caught during the *shutdown* system call.

SEE ALSO

connect(2B), socket(2B).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

socket - create an endpoint for communication

SYNOPSIS

int socket (domain, type, protocol)
int domain, type, protocol;

DESCRIPTION

socket creates an endpoint for communication and returns a descriptor.

The *domain* parameter specifies a communications domain within which communication will occur; this selects the protocol family that should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file **<sys/socket.h>**. The currently understood formats are as follows:

AF_UNIX	(UNIX internal protocols)
AF_INET	(ARPA Internet protocols)
AF_NS	(Xerox Network Systems protocols)

The socket has the indicated *type*, which specifies the semantics of communication. Currently defined types are as follows:

SOCK_STREAM

SOCK_DGRAM

A SOCK_STREAM type provides sequenced, reliable, two-way connection-based byte streams. An out-of-band data transmission mechanism may be supported. A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length).

The *protocol* specifies a protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, many protocols may exist. In this case, a protocol must be specified in this manner. The protocol number to use is particular to the communication domain in which communication is to occur (see *protocols*(4)).

Sockets of type SOCK_STREAM are full-duplex byte streams similar to pipes. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a *connect*(2B) call. Once connected, data may be transferred using *read*(2) and *write*(2) calls, the *readv*(2B) and *writev*(2B) calls, or some variant of the *send*(2B) and *recv*(2B) calls. When a session has been completed, a *close*(2) may be performed. Out-of-band data may also be transmitted as described in *send*(2B) and received as described in *recv*(2B).

The communications protocols used to implement a SOCK_STREAM ensure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, the connection is considered broken. Also, calls will indicate an error with -1 returned and with ETIMEDOUT as the specific code in the global variable errno. The protocols optionally keep sockets "warm" by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for an extended period (such as five minutes). A SIGPIPE signal is raised if a process sends on a broken stream; this causes naive processes that do not handle the signal to exit.

SOCK_DGRAM sockets allow datagrams to be sent to correspondents named in *send*(2B) calls. Datagrams are generally received with *recvfrom*(2B), which returns the next datagram with its return address.

fcntl(2) can be used to specify a process group to receive a SIGURG signal when the out-of-band data arrives. It may also enable nonblocking I/O and asynchronous notification of I/O events via the SIGIO signal.

The operation of sockets is controlled by socket-level *options*. These options are defined in the file **<sys/socket.h>**. *setsockopt*(2B) and *getsockopt*(2B) are used to set and get options, respectively.

socket will fail if one or more of the following is true:

[EMFILE]	The per-process descriptor table is full.
[ENFILE]	The system file table is full.
[ENOBUFS]	The resources to support the connection are not available.
[EPROTONOSUPPORT]	The specified domain, type or protocol is not supported.
[ESOCKTNOSUPPORT]	The socket type is not supported for the specified domain.
[ENXIO]	The system cannot access the specified device.
[EIO]	The protocol was not initialized.
[EINTR]	A signal was caught during the *socket* system call.

SEE ALSO

read(2), write(2), accept(2B), bind(2B), connect(2B), getsockname(2B), getsockopt(2B), listen(2B), readv(2B), recv(2B), select(2B), send(2B), shutdown(2B), socketpair(2B), writev(2B).
"Introductory Socket Tutorial", "Advanced Socket Tutorial" in the *CLIX System Guide*.

DIAGNOSTICS

Upon successful completion, a descriptor referencing the socket is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

socketpair - create a pair of connected sockets

SYNOPSIS

```
int socketpair (d, type, protocol, sv)
int d, type, protocol;
int sv[2];
```

DESCRIPTION

socketpair creates an unnamed pair of connected sockets in the specified domain *d*, of the specified type *type*, and using the optionally specified protocol *protocol* (see *socket*(2B)). The descriptors used in referencing the new sockets are returned in *sv*[0] and *sv*[1]. The two sockets are indistinguishable.

socketpair will fail if one or more of the following is true:

[EMFILE]	The per-process descriptor table is full.
[ENFILE]	The system file table is full.
[ENOBUFS]	The resources to support this connection are not available.
[EPROTONOSUPPORT]	The specified domain, type, or protocol is not supported.
[ESOCKTNOSUPPORT]	The socket type is not supported for the specified domain.
[ENXIO]	The system cannot access the specified device.
[EFAULT]	The address *sv* does not specify a valid part of the process address space.
[EINTR]	A signal was caught during the *socketpair* system call.

SEE ALSO

readv(2B), writev(2B), socket(2B).
pipe(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

symlink - make a symbolic link to a file

SYNOPSIS

int symlink (name1, name2)
char *name1, *name2;

DESCRIPTION

Name1 and *name2* point to path names naming files.

A symbolic link *name2* is created to *name1*. (*Name2* is the name of the file created; *name1* is the string used in creating the symbolic link.) Either name may be an arbitrary path name; the files need not be on the same file system.

symlink will fail if one or more of the following is true:

[ENOTDIR]	A component of the *name2* prefix is not a directory.
[ENOENT]	The named file does not exist or too many symbolic links were in the path of *name2*.
[EACCES]	Search permission is denied for a component of the *name2* prefix.
[EFAULT]	*Name1* or *name2* points to an invalid address.
[EINTR]	A signal was caught during the *symlink* system call.
[ENOLINK]	*Name2* points to a remote machine and the link to that machine is no longer active.
[EMULTIHOP]	Components of *name2* require hopping to multiple remote machines.

SEE ALSO

ln(1).
link(2), unlink(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

vfork - spawn a new process in a virtual memory efficient way

SYNOPSIS

int vfork ()

DESCRIPTION

vfork can be used to create new processes without fully copying the address space of the old process, which is inefficient in a paged environment. It is useful when the purpose of *fork*(2) would have been to create a new system context for an *execve*(2). *vfork* differs from *fork*(2) in that the child borrows the parent's memory and thread of control until a call to *execve*(2) or an exit (either by a call to *exit*(2) or abnormally). The parent process is suspended while the child is using its resources.

vfork returns 0 in the child's context and (later) the process ID of the child in the parent's context.

vfork can normally be used as *fork*(2). However, returning while running in the childs context from the procedure that called *vfork* does not work since the eventual return from *vfork* returns to a no longer existent stack frame. If *execve*(2) cannot be called, call *_exit*(2) rather than *exit*(2) since *exit*(2) will flush and close standard I/O channels and thereby corrupt the parent process's standard I/O data structures. (Even with *fork*(2) it is wrong to call *exit*(2) since buffered data would then be flushed twice.)

vfork will fail if the following is true:

[EAGAIN]	The system-imposed limit on the total number of processes would be exceeded, the system-imposed limit on the total number of processes per user would be exceeded, or the total amount of system memory available when reading via raw I/O, is temporarily insufficient.

SEE ALSO

exit(2), wait2(2I).
fork(2), exec(2), wait(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned to the child process and the process ID of the child process to the parent process. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

BUGS

To avoid a possible deadlock situation, processes that are children in the middle of a *vfork* are never sent SIGTTOU or SIGTTIN signals.

NAME

wait3 - wait for process to terminate

SYNOPSIS

```
#include <sys/wait.h>
#include <sys/time.h>
#include <sys/resource.h>

int wait3 (stat_loc, options, rusage)
union wait *stat_loc;
int options;
struct rusage *rusage;
```

DESCRIPTION

wait3 is implemented as *wait2*(2I).

SEE ALSO

wait2(2I).

CAVEATS

The *rusage* argument is not used in this implementation.

NAME

writev - write output to a socket

SYNOPSIS

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

int writev (s, iov, iovcnt)
int s, iovcnt;
struct iovec *iov;
```

DESCRIPTION

writev attempts to write data to the object referenced by the descriptor *s*. The output data is gathered from the *iovcnt* buffers specified by the members of the *iov* array: iov[0], iov[1], ..., iov[*iovcnt*−1].

The *iovec* structure is defined as

```
struct iovec {
        caddr_t    iov_base;
        int        iov_len;
};
```

Each *iovec* entry specifies the base address and length of an area in memory from which data should be written. *writev* will always write a complete area before proceeding to the next.

When using nonblocking I/O, *writev* may write fewer bytes than requested; the return value must be noted and the remainder of the operation should be retried when possible.

writev will fail if one or more of the following is true:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	The descriptor references a file, not a socket.
[EWOULDBLOCK]	The socket is marked nonblocking and the requested operation would block.
[EINTR]	A signal was caught during the *writev* system call.
[EFAULT]	An invalid user space address was specified.
[EPIPE]	The socket was disconnected or shut down for writing.
[EINVAL]	The maximum number of scatter gather locations was exceeded.
[ENOTCONN]	An attempt to send data on a stream socket that is not connected was made.
[EDESTADDRREQ]	An attempt to send data on a datagram socket without a destination address was made.
[EISCONN]	An attempt to perform a *sendto* on a connected socket was made.

SEE ALSO

fcntl(2), recv(2B), writev(2B), select(2B), getsockopt(2B), socket(2B).

DIAGNOSTICS

Upon successful completion, the number of bytes written is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

CAVEATS

writev is only supported on sockets, not on regular files.

21

NAME

exedata - setup for code execution in the process data section

SYNOPSIS

void exedata ()

DESCRIPTION

The CLIPPER architecture requires that certain "housekeeping" functions be performed in order to execute code that has been built in a writable area of the process address space (data, stack, or shared memory). *exedata* performs the necessary system functions to assure that the newly generated or modified code will execute correctly.

The proper order of events for execution of run-time generated code is listed below:

1) The code is generated in a writable area of the process address space.
2) The process calls *exedata* to assure proper execution of the new code.
3) Program control is given to the new code.

exedata should be called once after modification or generation of code and not each time the code is executed.

DIAGNOSTICS

No possible errors can occur.

NAME

getcpuid - return CLIPPER processor identifier

SYNOPSIS

int getcpuid (id)
int *id;

DESCRIPTION

getcpuid returns the CPU processor identifier from the CLIPPER system status word.

Current values for processor IDs are as follows:

Value	CLIPPER
0	C100
1	C200
2	C300

getcpuid will fail if the following is true:

[EFAULT] *Id* points to a nonwritable memory address.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

getmemsize, getfreemem, getavailsmem - return memory information

SYNOPSIS

int getmemsize ()
int getfreemem ()
int getavailsmem ()

DESCRIPTION

getmemsize returns the size of physical memory in bytes. *getfreemem* returns the amount of unused memory in bytes. *getavailsmem* returns the amount of memory that can be swapped in bytes.

NAME

kbmap - change the keyboard layout

SYNOPSIS

int kbmap (mapdata)
short mapdata[48][8];

DESCRIPTION

kbmap changes the definitions of certain keys on the keyboard. Only the letter, number, and punctuation keys on the main section of the keyboard are affected. The keypad and function keys are not changed.

Four qualifiers may be used with each key to change the character that is generated. The following eight combinations are valid:

unshifted keys
unshifted control keys
unshifted alternate keys
unshifted caps lock keys
shifted keys
shifted control keys
shifted alternate keys
shifted caps lock keys

Mapdata is an array of values specifying the value each key, both by itself and with qualifiers, will generate. The different values for each key must be specified in the order listed above. The keys must be listed in the following order, which is based on each key's physical location on the keyboard (identified using North American keycaps):

0	`
...	
12	=
13	Q
...	
24	]
25	A
...	
36	\
37	<
...	
47	/

kbmap will fail if the following is true:

[EFAULT] *Mapdata* points to an area outside of the users address space.

EXAMPLES

Mapdata for the North American key map is as follows:

```
unsigned char mapdata[ 48 ][ 8 ] = {
        { '`', '`', 0xe0, '`', '~', '~', 0xe1, '~' },
        { '1', '1', 0xe2, '1', '!', '!', 0xe3, '!' },
        { '2', '2', 0xe4, 0x00, '@', '@', 0xe5, 0x00 },
        { '3', '3', 0xe6, '3', '#', '#', 0xe7, '#' },
        ...
        { 'i', 'I', 0xd4, 0x09, 'I', 'I', 0xd5, 0x09 },
        { 'o', 'O', 0xd6, 0x0f, 'O', 'O', 0xd7, 0x0f },
        { 'p', 'P', 0xd8, 0x10, 'P', 'P', 0xd8, 0x10 },
        { '[', '[', 0xda, 0x1b, '{', '{', 0xd9, 0x1b },
        { ']', ']', 0xdc, 0x1d, '}', '}', 0xdb, 0x1d },
        { 'a', 'A', 0x9d, 0x01, 'A', 'A', 0x9e, 0x01 },
        ...
        { 'm', 'M', 0x95, 0x0d, 'M', 'M', 0x96, 0x0d },
        { ',', ',', 0x97, ',', ',', ',', 0x98, ',' },
        { '.', '.', 0x99, '.', '.', '.', 0x9a, '.' },
        { '/', '/', 0x9b, '/', '?', '?', 0x9c, '?' } };
```

SEE ALSO

kbmap(1), kbmap(4).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

readinfo - read system activity information

SYNOPSIS

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

int readinfo (itype, buf, nbytes)
int itype;
char *buf;
int nbytes;
```

DESCRIPTION

readinfo obtains system activity information. *Itype* specifies the type of information to be obtained. *Buf* is a pointer to the buffer where the data will be placed. *Nbytes* is the number of bytes of data to be copied.

The include file **<sys/sysinfo.h>** contains the following definitions that describe the types of information that may be obtained.

R_MINFO	Obtain system memory information.
R_SYSINFO	Obtain system time information.
R_SYSWAIT	Obtain system wait times.
R_DINFO	Obtain system disk activity.
R_RCINFO	Obtain Remote File Sharing (RFS) information.
R_SHLBINFO	Obtain shared library information.
R_SYSERR	Obtain system error statistics.

The amount of information available for each type is the size of the corresponding structure also defined in **<sys/sysinfo.h>**.

readinfo will fail if one or more of the following are true:

[EFAULT]	*Buf* points to a nonwritable memory address.
[EINVAL]	The type of information requested is not valid.

DIAGNOSTICS

Upon successful completion, the number of bytes read is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

setnodename - set new node name

SYNOPSIS

int setnodename (s)
char *s;

DESCRIPTION

setnodename sets the node name of the system as known by the communications network to the character string pointed to by *s*. This change is effective only until the next boot. *setnodename* will truncate *s* to a maximum length of eight bytes.

setnodename will fail if any of the following is true:

[EPERM]	The effective user ID is not super-user.
[EFAULT]	*S* points outside the allocated address space of the process.
[EINVAL]	The size of *s* exceeds the maximum number of characters allowed for a node name.

The current node name can be obtained using *uname*(2).

SEE ALSO

uname(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

sigcld - modify SIGCLD on stop signal option

SYNOPSIS

```
int sigcld (flag)
int flag;
```

DESCRIPTION

sigcld allows the caller to specify whether a SIGCLD signal is sent to its parent process if the child is stopped via a stop signal (SIGSTOP, SIGTSTP, SIGTTIN or SIGTTOU).

If *flag* is nonzero, the current process sends a SIGCLD signal to its parent process when stopped via a stop signal. Otherwise, a SIGCLD signal is not sent to its parent process when stopped via a stop signal.

Flag will be inherited by all children spawned by this process (see *fork*(2)) or new processes (see *exec*(2)).

SEE ALSO

signal(2), sigset(2).
fork(2), exec(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

sigcld returns the previous value of *flag*. *sigcld* cannot fail.

NAME

swap - swap space control

SYNOPSIS

#include <sys/swap.h>

int swap (si)
swpi_t *si;

DESCRIPTION

swap provides a mechanism to add a device to the system swap table area, remove a device from the system swap table, or obtain a list of the system swap table.

Si points to a structure with the following members:

```
char   si_cmd;      /* command code */
char   *si_buf;     /* pointer to a buffer or path name */
int    si_swplo;    /* first block number of the file */
int    si_nblks;    /* size of swap file in blocks */
```

Si_cmd contains one of the listed commands:

SI_LIST	Obtain a list of the system swap table. *Si_buf* points to a buffer with **sizeof**(*swpi_t*)*MSFILES bytes.
SI_ADD	Add a swap device to the system swap table. *Si_buf* points to the path name of the swap device. *Si_swplo* is the first block of the swap device, and *si_nblks* is the size of the swap device specified in blocks.
SI_DEL	Remove a swap device from the system swap table. *Si_buf* points to the path name of the swap device. *Si_swplo* is the first block of the swap device.

swap will fail if one or more of the following are true:

[EFAULT]	*Si* points to a location outside the allocated address space or some part of the buffer pointed to by the SI_LIST command's *si_buf* is outside the process's allocated space.
[EPERM]	The current user is not the super-user.
[ENOENT]	The swap file to be added or removed is invalid.
[ENOTBLK]	The swap device is not a block special file.
[EEXIST]	The swap device to be added is already in the system swap table.
[EINVAL]	*Si_cmd* contains an unknown command, or the swap device to be removed is not in the system swap table.
[ENOSPC]	The system swap table is full.

[ENOMEM] The user attempts to remove the last swap device or requests the removal of a swap device, which would leave the system with fewer swap pages than the minimum required.

SEE ALSO

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

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

sysid - get the system hardware identification number

SYNOPSIS

int sysid (id)
char id[6];

DESCRIPTION

sysid fills the array *id* with the system's hardware identification number. The system ID is of the form

08-00-36-*xx*-*yy*-*zz*

where *xx*, *yy*, and *zz* are unique for each system. *sysid* fills the array *id* as listed below.

Byte	Value
0	08
1	00
2	36
3	*xx*
4	*yy*
5	*zz*

sysid will fail if the following is true:

[EFAULT]	*Id* points to a nonwritable location in the address space.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

ucpclr - clear process UCP priority

SYNOPSIS

```
int ucpclr (pid)
int pid;
```

DESCRIPTION

ucpclr clears the User Controlled Priority (UCP) attribute of the process with process ID *pid*. If *pid* has a value of 0, the calling process is assumed. If no UCP attribute is set, the call is ignored. This has no effect on the signal mask, which may have been set with *ucpsig*(2I).

ucpclr will fail if the following is true:

[ESRCH]	Process ID *pid* does not exist.

SEE ALSO

ucpsig(2I), ucprelse(2I), ucppri(2I), ucpinq(2I), ucpset(2I), ucpnice(1).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

ucpinq - return the UCP priority

SYNOPSIS

```
int ucpinq (pid)
int pid;
```

DESCRIPTION

ucpinq returns the User Controlled Priority (UCP) of the process with process id *pid*. If *pid* is 0, the priority of the calling process is returned. If the process is not running at a UCP priority, 128 is returned.

ucpinq will fail if the following is true:

[ESRCH]	No process exists with process ID *pid*.

SEE ALSO

ucpsig(2I), ucprelse(2I), ucppri(2I), ucpclr(2I), ucpset(2I).

DIAGNOSTICS

If the process exists and is a *ucp* process, the *ucp* priority of the process is returned. If the process exists but is not a UCP process, 128 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

ucppri - check if a UCP priority is already in use

SYNOPSIS

```
int ucppri (pri)
int pri;
```

DESCRIPTION

ucppri determines if any process is set to the User Controlled Priority (UCP) *pri*. A value of 1 is returned if any process is at *pri*. Otherwise, 0 is returned.

SEE ALSO

ucpsig(2I), ucprelse(2I), ucpinq(2I), ucpclr(2I), ucpset(2I).

DIAGNOSTICS

No errors can occur.

CAVEATS

There is no way to determine if a signal routine of another process may be set to UCP priority *pri*.

NAME

ucprelse - reset a process's priority after handling a signal

SYNOPSIS

void ucprelse ()

DESCRIPTION

ucprelse resets the calling processes User Controlled Priority (UCP) to the priority it was running at before handling a signal (see *ucpsig*(2I) for exceptions).

SEE ALSO

ucpsig(2I), ucpclr(2I), ucppri(2I), ucpinq(2I), ucpset(2I).

DIAGNOSTICS

No possible errors can occur.

NOTES

ucprelse should be called when completing a *signal*(2) handling routine. It does not need to be called if *sigset*(2) is used. *sigset*(2) will perform a *ucprelse* as part of the *sigrelse*(2).

NAME

ucpset - set a process to a UCP priority

SYNOPSIS

#include <sys/ucp.h>

int ucpset (pid, pri, flag)
int pid, pri, flag;

DESCRIPTION

ucpset sets the process with process ID *pid* to the User Controlled Priority (UCP) priority *pri*. If *pid* has a value of 0, the calling process is changed. A maximum *pri* value of 127 and a minimum *pri* value of 0 are imposed by the system. The process must have super-user privileges.

A process at UCP priority 0 will be run before any other UCP process and before any other non-UCP user process. The time sharing between two processes running at the same UCP priority is not defined.

Flag determines whether the UCP priority is carried over to child processes. If *flag* is set to UCP_FLAG, the UCP priority is not carried over to any child processes.

ucpset will fail if one or more of the following are true:

[EPERM]	The effective user ID of the calling process is not super-user.
[EINVAL]	The *pri* value was greater than 127 or less than 0.
[ESRCH]	Process ID *pid* does not exist.

SEE ALSO

ucpsig(2I), ucprelse(2I), ucpinq(2I), ucppri(2I), ucpclr(2I), ucpnice(1).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NAME

ucpsig - set process to a UCP priority on receipt of a signal

SYNOPSIS

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

int ucpsig (pri, flag, sigmask)
int  pri, flag;
long sigmask;
```

DESCRIPTION

Sigmask specifies a set of signals that causes the calling process to run at User Controlled Priority (UCP) *pri* when handling these signals. *Sigmask* is created by setting the bit corresponding to the desired signal, (1L << ((signum) - 1)). The process is returned to its previous priority when either *sigrelse*(2) or *ucprelse*(2I) is called.

Clearing a signal that was previously set in *sigmask* is accomplished by calling *ucpsig* with a different *sigmask*.

The calling process must have super-user privileges. A maximum *pri* value of 127 and a minimum of 0 are imposed by the system.

Flag determines whether the UCP priority is carried over to child processes. If *flag* is set to UCP_FLAG, the UCP priority is not carried over.

ucpsig will fail if one or more of the following are true:

[EPERM]	The effective user ID of the calling process in not super-user.
[EINVAL]	The *pri* value was greater than 127 or less than 0.

SEE ALSO

ucprelse(2I), ucppri(2I), ucpinq(2I), ucpnice(1), ucpset(2I), ucpclr(2I), sigset(2), signal(2).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NOTES

If *ucpset*(2I) is called from within a signal-handling routine, the process will finish handling the signal at the new priority and upon release will maintain the new priority. (It will not return to the priority it was running at before receiving the signal.)

If *ucpclr*(2I) is called from within a signal-handling routine, the process will finish handling the signal at a non-UCP priority and will remain at that priority after the signal is released.

WARNINGS

It is recommended that *ucpsig* be used with *sigset*(2) and not with *signal*(2).

NAME

vlock - lock an area of memory

SYNOPSIS

int vlock (addr, length)
int addr, length;

DESCRIPTION

vlock is a page-locking mechanism. The area of memory to be locked starts at virtual address *addr* and is *length* bytes long. *vlock* returns a unique key used to identify the area to be unlocked by *vunlock*(2I).

vlock will fail if one or more of the following are true:

[EFAULT]	The range of addresses specified are not valid.
[EAGAIN]	The necessary resources are not available at this time.

SEE ALSO

vunlock(2I).

DIAGNOSTICS

Upon successful completion, a unique key is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

CAVEATS

Addr cannot be 0.

NAME

vunlock - unlock an area of memory

SYNOPSIS

int vunlock (key, flag)
int key, flag;

DESCRIPTION

vunlock is the page-unlocking mechanism for memory locked with *vlock*(2I). The *key* is a unique identifier returned by *vlock*(2I).

If *flag* is set, the page reference and modify bits are cleared for all pages contained fully in the address range. This action will invalidate any data associated with these pages.

vunlock will fail if one or more of the following are true:

[EINVAL]	The *key* is not a valid key.
[EPERM]	The process attempting to unlock the area is not the same process that locked the area.

SEE ALSO

vlock(2I).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

WARNINGS

Any pages within a shared memory region that were locked by *vlock*(2I) must be unlocked before the region is detached.

NAME

wait2 - wait for process to terminate

SYNOPSIS

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

int wait2 (stat_loc, options)
union wait *stat_loc;
int options;
```

DESCRIPTION

wait2 provides an alternate interface for programs that must not block when collecting the status of child processes. The *options* parameter indicates that the call should not block if no processes wish to report status (WNOHANG), and/or that only children of the current process that are stopped due to a SIGTTIN, SIGTTOU, SIGTSTP, or SIGSTOP signal should have their statuses reported (WUNTRACED).

When the WNOHANG option is specified and no processes wish to report status, *wait2* returns a pid of 0. The WNOHANG and WUNTRACED options may be combined by ORing the two values.

If *stat_loc* is nonzero, 16 bits of information called status are stored in the low-order 16 bits of the location pointed to by *stat_loc*. *Stat_loc* can be used to differentiate between stopped and terminated child processes. If the child process is terminated, *stat_loc* identifies the cause of termination and passes useful information to the parent. This is accomplished in the following manner:

> If the child process is stopped, the high-order eight bits of the status will contain the number of the signal that caused the process to stop and the low-order eight bits will be set equal to 0177.
>
> If the child process terminated due to an *exit*(2) call, the low-order eight bits of status will be zero and the high-order eight bits will contain the low-order eight bits of the argument that the child process passed to *exit*(2).
>
> If the child process terminated due to a signal, the high-order eight bits of status will be zero and the low-order eight bits will contain the number of the signal that caused the termination. In addition, if the low-order seventh bit (bit 200) is set, a *core*(4) image file will have been produced (see *signal*(2)).

If the parent process terminates without waiting for its child processes to terminate, the parent process ID of each child process is set to 1. This means the initialization process inherits the child processes (see *intro*(2)).

wait2 will fail and return immediately if one or more of the following is true:

[ECHILD]	The calling process has no existing unwaited-for child processes.

[EFAULT] The *stat_loc* argument points to an illegal address.

SEE ALSO

sigcld(2I), signal(2), exit(2).
wait(2), fork(2), pause(2), ptrace(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

wait2 returns -1 if there are no children not previously waited for; 0 is returned if WNOHANG is specified and no stopped or exited children exist.

If *wait2* returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

Library Routines (3)

NAME

intro - introduction to functions and libraries

DESCRIPTION

This section describes functions found in various libraries, other than those that directly invoke CLIX system primitives. The system primitives are described in Sections (2), (2B), and (2I) of this volume. Certain major collections are identified by a letter after the section number:

(3C) These functions, those of Section (2), and those marked (3S) constitute the Standard C Library *libc*, that is automatically loaded by the C compiler, *cc*(1). (For this reason, the (3C) and (3S) sections compose one section of this manual.) The link editor *ld*(1) searches this library under the **-lc** option. A shared library version of *libc* can be searched using the **-lc_s** option, resulting in smaller *a.out*(4) files. Declarations for some of these functions may be obtained from #include files indicated on the appropriate pages.

(3S) These functions constitute the "standard I/O package" (see *stdio*(3S)). These functions are in the library *libc*, mentioned above. Declarations for these functions may be obtained from the #include file **<stdio.h>**.

(3B) These functions, those of Section (2B), and those marked (3R) constitute the Berkeley Software Distribution (BSD) Library, *libbsd*. They are not automatically loaded by the C compiler, *cc*(1); however, the link editor *ld*(1) searches this library under the **-lbsd** option.

(3N) These functions constitute the Intergraph Network Library, *libinc*. They are not automatically loaded by the C compiler, *cc*(1); however, the link editor *ld*(1) searches this library under the **-linc** option.

(3R) These functions constitute the Remote Procedure Call (RPC) and External Data Representation (XDR) package. These functions are in the library, *libbsd*, mentioned above.

(3A) These functions and those of Section (2I) constitute the Intergraph Extensions Library, *libix*. These functions compose the asynchronous I/O package. They are not automatically loaded by the C compiler, *cc*(1); however, the link editor, *ld*(1) searches this library under the **-lix** option.

(3F) These functions constitute the Intergraph extensions to the FORTRAN intrinsic function library, *libf*. These functions are automatically available to the FORTRAN programmer and require no special invocation of the compiler.

FILES

$LIBDIR	usually **/lib**
$LIBDIR/libc.a	
$LIBDIR/libc_s.a	
/shlib/libc_s	

/usr/lib/libbsd.a
/usr/lib/libinc.a
/usr/lib/libix.a

SEE ALSO

cc(1), ld(1), intro(2).
ar(1), lint(1), nm(1), stdio(3S) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Functions in the C Library (3C) may return the conventional values 0 or ±HUGE (the largest-magnitude single-precision floating-point numbers; HUGE is defined in the **<math.h>** header file) when the function is undefined for the given arguments or when the value is not representable. In these cases, the external variable *errno* (see *intro*(2)) is set to the value EDOM or ERANGE.

WARNINGS

Many of the functions in the libraries call and/or refer to other functions and external variables described in this section and in Section (2) (*System Calls*). If a program inadvertently defines a function or external variable with the same name, the presumed library version of the function or external variable may not be loaded. The *lint*(1) program checker reports name conflicts of this kind as "multiple declarations" of the names in question. Definitions for Sections (2), (3C), and (3S) are checked automatically. Other definitions can be included by using the -l option. Using *lint*(1) is highly recommended.

3C & 3S

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent - get group file entry

SYNOPSIS

```
#include <grp.h>

struct group *getgrent ()

struct group *getgrgid (gid)
int gid;

struct group *getgrnam (name)
char *name;

void setgrent ()

void endgrent ()

struct group *fgetgrent (f)
FILE *f;
```

DESCRIPTION

getgrent, *getgrgid*, and *getgrnam* return pointers to an object with the following structure containing the broken-out fields of a line in the **/etc/group** file. Each line contains a *group* structure, defined in the **<grp.h>** header file.

```
struct group {
        char    *gr_name;   /* the name of the group */
        char    *gr_passwd; /* the encrypted group password */
        int     gr_gid;     /* the numerical group ID */
        char    **gr_mem;   /* vector of pointers to member names */
};
```

The CLIX implementation of these routines includes support for the Yellow Pages (see *ypserv*(1M) for more information).

When first called, *getgrent* returns a pointer to the first group structure in the file. Thereafter, it returns a pointer to the next group structure in the file; so, successive calls may be used to search the entire file. *getgrgid* searches from the beginning of the file until a numerical group id matching *gid* is found and returns a pointer to the structure in which it was found. *getgrnam* searches from the beginning of the file until a group name matching *name* is found and returns a pointer to the structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a null pointer.

A call to *setgrent* has the effect of rewinding the group file to allow repeated searches. *endgrent* may be called to close the group file when processing is complete.

fgetgrent returns a pointer to the next group structure in the stream *f*. The stream *f* should match the format of **/etc/group**.

FILES

/etc/group

SEE ALSO

getpwent(3C), group(4).
ypserv(1M) in the *CLIX System Administrator's Reference Manual.*
getlogin(3C) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

A null pointer is returned on EOF or error.

WARNINGS

The above routines use **<stdio.h>**, which causes them to increase the size of programs not otherwise using standard I/O more than expected.

CAVEATS

All information is contained in a static area, so it must be copied if it is to be saved.

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent - get password file entry

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent ()

struct passwd *getpwuid (uid)
int uid;

struct passwd *getpwnam (name)
char *name;

void setpwent ()

void endpwent ()

struct passwd *fgetpwent (f)
FILE *f;
```

DESCRIPTION

getpwent, *getpwuid*, *getpwnam* return a pointer to an object with the following structure containing the broken-out fields of a line in the **/etc/passwd** file. Each line in the file contains a *passwd* structure declared in the **<pwd.h>** header file:

```
struct passwd {
        char    *pw_name;
        char    *pw_passwd;
        int     pw_uid;
        int     pw_gid;
        char    *pw_age;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};
```

The fields are described in *passwd*(4).

The CLIX implementation of these routines includes support for the Yellow Pages (see *ypserv*(1M) for more information).

When first called, *getpwent* returns a pointer to the first passwd structure in the file. Thereafter, it returns a pointer to the next passwd structure in the file; so, successive calls can be used to search the entire file. *getpwuid* searches from the beginning of the file until a numerical user ID matching *uid* is found and returns a pointer to the structure in which it was found. *getpwnam* searches from the beginning of the file until a login name matching *name* is found and returns a pointer to the structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a null pointer.

A call to *setpwent* has the effect of rewinding the password file to allow repeated searches. *endpwent* may be called to close the password file when processing is complete.

fgetpwent returns a pointer to the next passwd structure in the stream *f*. The stream *f* should match the format of **/etc/passwd**.

FILES

/etc/passwd

SEE ALSO

getgrent(3C), passwd(4).
ypserv(1M) in the *CLIX System Administrator's Reference Manual.*
getlogin(3C) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

A null pointer is returned on EOF or error.

WARNINGS

The above routines use **<stdio.h>**, which causes them to increase the size of programs not otherwise using standard I/O more than might be expected.

CAVEATS

All information is contained in a static area, so it must be copied if it is to be saved.

3B

NAME

intro - introduction to BSD library functions

DESCRIPTION

This section describes functions found in the Berkeley Software Distribution (BSD) library, **libbsd.a**, that do not directly invoke CLIX system primitives. The link editor *ld*(1) searches this library under the **-lbsd** option.

List Of Functions

Name	Appears on Page	Description
bcmp	bstring(3B)	bit and byte string operations
bcopy	bstring(3B)	bit and byte string operations
bzero	bstring(3B)	bit and byte string operations
dbm_clearerr	ndbm(3B)	database subroutines
dbm_close	ndbm(3B)	database subroutines
dbm_delete	ndbm(3B)	database subroutines
dbm_error	ndbm(3B)	database subroutines
dbm_fetch	ndbm(3B)	database subroutines
dbm_firstkey	ndbm(3B)	database subroutines
dbm_nextkey	ndbm(3B)	database subroutines
dbm_open	ndbm(3B)	database subroutines
dbm_store	ndbm(3B)	database subroutines
endhostent	gethostbyname(3B)	get network host entry
endnetent	getnetent(3B)	get network entry
endprotoent	getprotoent(3B)	get protocol entry
endservent	getservent(3B)	get service entry
ffs	bstring(3B)	bit and byte string operations
gethostbyaddr	gethostbyname(3B)	get network host entry
gethostbyname	gethostbyname(3B)	get network host entry
gethostent	gethostbyname(3B)	get network host entry
getnetbyaddr	getnetent(3B)	get network entry
getnetbyname	getnetent(3B)	get network entry
getnetent	getnetent(3B)	get network entry
getprotobyname	getprotoent(3B)	get protocol entry
getprotobynumber	getprotoent(3B)	get protocol entry
getprotoent	getprotoent(3B)	get protocol entry
getservbyname	getservent(3B)	get service entry
getservbyport	getservent(3B)	get service entry
getservent	getservent(3B)	get service entry
htonl	byteorder(3B)	convert values between host and network byte order
htons	byteorder(3B)	convert values between host and network byte order
index	string(3B)	string operations
inet_addr	inet(3B)	Internet address manipulation routines

Name	Appears on Page	Description
inet_lnaof	inet(3B)	Internet address manipulation routines
inet_makeaddr	inet(3B)	Internet address manipulation routines
inet_netof	inet(3B)	Internet address manipulation routines
inet_network	inet(3B)	Internet address manipulation routines
insque	insque(3B)	insert/remove element from a queue
ntohl	byteorder(3B)	convert values between host and network byte order
ntohs	byteorder(3B)	convert values between host and network byte order
random	random(3B)	better random number generator
rcmd	rcmd(3B)	routines for returning a stream to a remote command
remque	insque(3B)	insert/remove element from a queue
rexec	rexec(3B)	return stream to a remote command
rindex	string(3B)	string operations
rresvport	rcmd(3B)	routines for returning a stream to a remote command
ruserok	rcmd(3B)	routines for returning a stream to a remote command
sethostent	gethostbyname(3B)	get network host entry
setnetent	getnetent(3B)	get network entry
setprotoent	getprotoent(3B)	get protocol entry
setservent	getservent(3B)	get service entry
srandom	random(3B)	better random number generator

FILES

/lib/libbsd.a	the BSD library

SEE ALSO

intro(2B), ld(1).

NAME

bstring: bcopy, bcmp, bzero, ffs - bit and byte string operations

SYNOPSIS

```
int bcopy (src, dst, length)
char *src, *dst;
int length;

int bcmp (b1, b2, length)
char *b1, *b2;
int length;

int bzero (b, length)
char *b;
int length;

int ffs (i)
int i;
```

DESCRIPTION

The functions *bcopy*, *bcmp*, and *bzero* operate on variable length strings. They do not check for null bytes as do the routines in *string*(3C).

bcopy copies *length* bytes from string *src* to the string *dst*.

bcmp compares byte string *b1* to byte string *b2*, returning zero if they are identical, nonzero otherwise. Both strings are assumed to be *length* bytes long.

bzero places *length* 0 bytes in the string *b1*.

ffs finds the first bit set in the argument passed and returns the index of that bit. Bits are numbered starting with 1. A returned value of 0 indicates the value passed is zero.

NOTES

The *bcopy* routine reverses parameters from *strcpy*(3C).

NAME

htonl, htons, ntohl, ntohs - convert values between host and network byte order

SYNOPSIS

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

u_long htonl (hostlong)
u_long hostlong;

u_short htons (hostshort)
u_short hostshort;

u_long ntohl (netlong)
u_long netlong;

u_short ntohs (netshort)
u_short netshort;
```

DESCRIPTION

These routines convert 16- and 32-bit quantities between network and host byte order. These routines are most often used with Internet addresses and ports as returned by *gethostbyname*(3B) and *getservent*(3B).

SEE ALSO

gethostbyname(3B), getservent(3B).

NAME

gethostbyname, gethostbyaddr, gethostent, sethostent, endhostent - get network host entry

SYNOPSIS

```
#include <netdb.h>

extern int h_errno;

struct hostent *gethostbyname (name)
char *name;

struct hostent *gethostbyaddr (addr, len, type)
char *addr;
int len, type;

struct hostent *gethostent ()

void sethostent (stayopen)
int stayopen;

void endhostent ()
```

DESCRIPTION

gethostbyname and *gethostbyaddr* return a pointer to an object with the following structure. This structure contains information obtained from the broken-out fields from a line in the file **/etc/hosts**, or, if Yellow Pages (YP) is running, from an entry in the YP database (see *ypfiles*(4)).

```
struct hostent {
        char    *h_name;        /* official name of host */
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* host address type */
        int     h_length;       /* length of address */
        char    **h_addr_list;  /* list of addresses from name server */
};
#define h_addr h_addr_list[0] /* address, backward compatibility */
```

The members of this structure are as follows:

h_name	Official name of the host.
h_aliases	A zero-terminated array of alternate names for the host.
h_addrtype	The type of address being returned; currently always AF_INET.
h_length	The length, in bytes, of the address.
h_addr_list	A zero-terminated array of network addresses for the host. Host addresses are returned in network byte order.
h_addr	The first address in h_addr_list; this is for backward compatibility.

sethostent allows a request for the use of a connected socket using Transmission Control Protocol (TCP) for queries. A nonzero *stayopen* flag sets the option to send all queries to the name server using TCP and to retain the connection after each call to *gethostbyname* or *gethostbyaddr*.

endhostent closes the TCP connection.

gethostbyname and *gethostbyaddr* sequentially search from the beginning of the file until a matching host name or address and type is found or until EOF is encountered. Internet addresses may be obtained from character strings representing numbers expressed in the Internet standard "." notation with the routines described in *inet*(3B). If an address is supplied, the length of the address must also be supplied.

FILES

/etc/hosts

SEE ALSO

hosts(4).
namex(1M) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Error return status from *gethostbyname* and *gethostbyaddr* is indicated by the return of a null pointer. The external integer *h_errno* may then be checked to see whether this is a temporary failure or an invalid or unknown host.

h_errno can have the following values:

HOST_NOT_FOUND	No such host is known.
TRY_AGAIN	This is usually a temporary error and means that the local server did not receive a response from an authoritative server. A retry at some later time may succeed.
NO_RECOVERY	This is a nonrecoverable error.
NO_ADDRESS	The requested name is valid but does not have an IP address; this is not a temporary error. This means another type of request to the name server will result in an answer.

BUGS

All information is contained in a static area so it must be copied to be saved. Only the Internet address format is currently understood.

CAVEATS

gethostent reads the next line of **/etc/hosts**, opening the file if necessary.

sethostent is redefined to open and rewind the file. If the *stayopen* argument is nonzero, the hosts database will not be closed after each call to *gethostbyname* or *gethostbyaddr*. *endhostent* is redefined to close the file.

NAME

getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent - get network entry

SYNOPSIS

```
#include <netdb.h>

struct netent *getnetent ()

struct netent *getnetbyname (name)
char *name;

struct netent *getnetbyaddr (net, type)
long net;
int type;

void setnetent (stayopen)
int stayopen;

void endnetent ()
```

DESCRIPTION

getnetent, *getnetbyname*, and *getnetbyaddr* return a pointer to an object with the following structure containing the broken-down fields of a line in the network database, **/etc/networks.**

```
struct netent {
        char     *n_name;      /* official name of net */
        char     **n_aliases;  /* alias list */
        int      n_addrtype;   /* net number type */
        u_long   n_net;        /* net number */
};
```

The members of this structure are as follows:

n_name	The official name of the network.
n_aliases	A zero-terminated list of alternate names for the network.
n_addrtype	The type of the network number returned; currently only AF_INET.
n_net	The network number. Network numbers are returned in machine-byte order.

getnetent reads the next line of the file, opening the file if necessary.

setnetent opens and rewinds the file. If the *stayopen* flag is nonzero, the net database will not be closed after each call to *getnetbyname* or *getnetbyaddr*.

endnetent closes the file.

getnetbyname and *getnetbyaddr* sequentially search from the beginning of the file until a matching network name or address and type is found or until EOF is encountered. Network numbers are supplied in host order.

FILES

/etc/networks

SEE ALSO

networks(4).

DIAGNOSTICS

A null pointer (0) is returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied to be saved. Only Internet network numbers are currently understood. It is expected that network numbers will fit in 32 bits or less.

NAME

getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent - get protocol entry

SYNOPSIS

#include <netdb.h>

struct protoent *getprotoent ()

struct protoent *getprotobyname (name)
char *name;

struct protoent *getprotobynumber (proto)
int proto;

void setprotoent (stayopen)
int stayopen;

void endprotoent ()

DESCRIPTION

getprotoent, *getprotobyname*, and *getprotobynumber* return a pointer to an object with the following structure containing the broken-down fields of a line in the network protocol database, **/etc/protocols.**

```
struct protoent {
        char    *p_name;        /* official name of protocol */
        char    **p_aliases;    /* alias list */
        int     p_proto;        /* protocol number */
};
```

The members of this structure are as follows:

p_name	The official name of the protocol.
p_aliases	A zero-terminated list of alternate names for the protocol.
p_proto	The protocol number.

getprotoent reads the next line of the file, opening the file if necessary.

setprotoent opens and rewinds the file. If the *stayopen* flag is nonzero, the network database will not be closed after each call to *getprotobyname* or *getprotobynumber*.

endprotoent closes the file.

getprotobyname and *getprotobynumber* sequentially search from the beginning of the file until a matching protocol name or number is found or until EOF is encountered.

FILES

/etc/protocols

SEE ALSO

protocols(4).

DIAGNOSTICS

A null pointer (0) is returned at EOF or when an error occurs.

BUGS

All information is contained in a static area so it must be copied to be saved. Only the Internet protocols are currently understood.

NAME

getservent, getservbyport, getservbyname, setservent, endservent - get service entry

SYNOPSIS

```
#include <netdb.h>

struct servent *getservent ()

struct servent *getservbyname (name, proto)
char *name, *proto;

struct servent *getservbyport (port, proto)
int port;
char *proto;

void setservent (stayopen)
int stayopen

void endservent ()
```

DESCRIPTION

getservent, *getservbyname*, and *getservbyport* return a pointer to an object with the following structure containing the broken-down fields of a line in the network services database, **/etc/services.**

```
struct servent {
        char    *s_name;        /* official name of service */
        char    **s_aliases;    /* alias list */
        int     s_port;         /* port service resides at */
        char    *s_proto;       /* protocol to use */
};
```

The members of this structure are as follows:

s_name	The official name of the service.
s_aliases	A zero-terminated list of alternate names for the service.
s_port	The port number at which the service resides. Port numbers are returned in network byte order.
s_proto	The name of the protocol to use when contacting the service.

getservent reads the next line of the file, opening the file if necessary.

setservent opens and rewinds the file. If the *stayopen* flag is nonzero, the network database will not be closed after each call to *getservbyname* or *getservbyport*.

endservent closes the file.

getservbyname and *getservbyport* sequentially search from the beginning of the file until a matching protocol name or port number is found or until EOF is encountered. If a protocol name is also supplied (non-null), searches must match the protocol. Port numbers must be given in network byte order.

FILES

/etc/services

SEE ALSO

getprotoent(3B), services(4).

DIAGNOSTICS

A null pointer (0) is returned at EOF or when an error occurs.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. It is expected that port numbers will fit in a 32-bit quantity.

NAME

inet: inet_addr, inet_network, inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof - Internet address manipulation routines

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

unsigned long inet_addr (cp)
char *cp;

unsigned long inet_network (cp)
char *cp;

char *inet_ntoa (in)
struct in_addr in;

struct in_addr inet_makeaddr (net, lna)
int net, lna;

int inet_lnaof (in)
struct in_addr in;

int inet_netof (in)
struct in_addr in;
```

DESCRIPTION

The routines **inet_addr** and *inet_network* interpret character strings representing numbers expressed in the Internet standard "." notation, returning numbers suitable for use as Internet addresses and Internet network numbers, respectively.

The routine *inet_ntoa* takes an Internet address and returns an ASCII string representing the address in "." notation. The routine *inet_makeaddr* takes an Internet network number and a local network address and constructs an Internet address. The routines *inet_netof* and *inet_lnaof* break apart Internet host addresses, returning the network number and local network address part, respectively.

All Internet addresses are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

Internet Addresses

Values specified using the "." notation assume one of the following forms:

```
a.b.c.d
a.b.c
a.b
a
```

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on the CLIPPER, the bytes referred to above appear as d.c.b.a. CLIPPER bytes are ordered from right to left.

When a three-part address is specified, the last part is interpreted as a 16-bit quantity and placed in the rightmost two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as **128.***net.host*.

When a two-part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the rightmost three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as *lqnet.host*.

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied in the notation may be decimal, octal, or hexadecimal, as specified in the C language (i.e., a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

SEE ALSO

gethostbyname(3B), getnetent(3B), hosts(4), networks(4).

DIAGNOSTICS

A value of -1 is returned by *inet_addr* and *inet_network* for malformed requests.

BUGS

Host versus network byte ordering is confusing. A simple way to specify Class C network addresses similarly to that for Classes A and B is needed. The string returned by *inet_ntoa* resides in a static memory area.

inet_addr should return a struct *in_addr*.

NAME

insque, remque - insert/remove element from a queue

SYNOPSIS

void insque (elem, pred)
struct qelem *elem, *pred;

void remque (elem)
struct qelem *elem;

DESCRIPTION

insque and *remque* manipulate queues built from doubly-linked lists. Each element in the queue must be in the form of the structure *qelem* as follows:

```
struct qelem {
          struct   qelem *q_forw;
          struct   qelem *q_back;
          char     q_data[];
};
```

insque inserts *elem* in a queue immediately after *pred*; *remque* removes an entry *elem* from a queue.

DIAGNOSTICS

No possible errors can occur.

NAME

dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr - database subroutines

SYNOPSIS

```
#include <ndbm.h>

typedef struct {
        char   *dptr;
        int    dsize;
} datum;

DBM *dbm_open (file, flags, mode)
char *file;
int flags, mode;

void dbm_close (db)
DBM *db;

datum dbm_fetch (db, key)
DBM *db;
datum key;

int dbm_store (db, key, content, flags)
DBM *db;
datum key, content;
int flags;

int dbm_delete (db, key)
DBM *db;
datum key;

datum dbm_firstkey (db)
DBM *db;

datum dbm_nextkey (db)
DBM *db;

int dbm_error (db)
DBM *db;

int dbm_clearerr (db)
DBM *db;
```

DESCRIPTION

These functions maintain key/content pairs in a database. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two file system accesses.

Keys and *contents* are described by the *datum* typedef. A *datum* specifies a string of *dsize* bytes pointed to by *dptr*. Arbitrary binary data, as well as normal ASCII strings, are allowed. The database is stored in two files. One file is a directory containing a bit map and has ".dir" as its suffix. The second file contains all data and has ".pag" as its suffix.

Before a database can be accessed, it must be opened by *dbm_open*. This will open and/or create the files *file*.**dir** and *file*.**pag** depending on the flags parameter (see *open*(2)).

Once open, the data stored under a key is accessed by *dbm_fetch* and data is placed under a key by *dbm_store*. The *flags* field can be either *dbm_insert* or *dbm_replace*. *dbm_insert* will only insert new entries in to the database and will not change an existing entry with the same key. *dbm_replace* will replace an existing entry if it has the same key. A key (and its associated contents) is deleted by *dbm_delete*. A linear pass through all keys in a database may be made in an (apparently) random order by use of *dbm_firstkey* and *dbm_nextkey*. *dbm_firstkey* will return the first key in the database. *dbm_nextkey* will return the next key in the database. This code will traverse the database:

```
for (key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

dbm_error returns nonzero when an error has occurred reading or writing the database. *dbm_clearerr* resets the error condition on the named database.

SEE ALSO

open(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

All functions that return an integer indicate errors with negative values. A zero return indicates success. Routines that return a *datum* indicate errors with a null (0) *dptr*. If *dbm_store* called with a *flags* value of *dbm_insert* finds an existing entry with the same key, it returns 1.

BUGS

The ***.pag** file will contain holes so that its apparent size is approximately four times its actual content. These files cannot be copied by normal means (*cp*(1), *cat*(1), *tar*(1), *ar*(1)) without filling in the holes.

Dptr pointers returned by these subroutines point to static storage changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 4096 bytes). Moreover, all key/content pairs that hash together must fit on a single block. *dbm_store* will return an error if a disk block fills with inseparable data.

dbm_delete does not physically reclaim file space, although it makes it available for reuse.

The order of keys presented by *dbm_firstkey* and *dbm_nextkey* depends on a hashing function.

NAME

random, srandom - better random number generator

SYNOPSIS

long random ()

void srandom (seed)
int seed;

DESCRIPTION

random uses a nonlinear additive feedback random number generator. *random* employs a default table size of 31 long integers to return pseudo-random numbers in the range from 0 to 2^31−1. The period of this random number generator is approximately 16(2^31−1).

random and *srandom* have (almost) the same calling sequence and initialization properties as *rand*(3C)/*srand*(3C). The difference is that *rand*(3C) produces a much less random sequence. In fact, the low dozen bits generated by *rand*(3C) go through a cyclic pattern. All the bits generated by *random* are usable. For example, **random()&01** will produce a random binary value.

Unlike *srand*(3C), *srandom* does not return the old seed because the amount of state information used is much more than a single word. Like *rand*(3C), however, *random* will by default produce a sequence of numbers that can be duplicated by calling *srandom* with 1 as the seed.

With 256 bytes of state information, the period of the random number generator is greater than 2^69, which should be sufficient for most purposes.

SEE ALSO

rand(3C) in the *UNIX System V Programmer's Reference Manual.*

NOTES

random executes at approximately 2/3 the speed of *rand*(3C).

NAME

rcmd, rresvport, ruserok - routines for returning a stream to a remote command

SYNOPSIS

```
int rcmd (ahost, inport, locuser, remuser, cmd, fd2p)
char **ahost;
int inport;
char *locuser, *remuser, *cmd;
int *fd2p;

int rresvport (port)
int *port;

int ruserok (rhost, superuser, ruser, luser)
char *rhost;
int superuser;
char *ruser, *luser;
```

DESCRIPTION

rcmd is a routine used by the super-user to execute a command on a remote machine using an authentication scheme based on reserved port numbers. *rresvport* is a routine that returns a descriptor to a socket with an address in the privileged port space. *ruserok* is a routine used by servers to authenticate clients requesting service with *rcmd*. All three functions are in the same file and are used by the *rshd*(1M) server (among others).

rcmd looks up the host **ahost* using *gethostbyname*(3B), returning -1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host and a connection is established to a server residing at the Internet port *inport*.

If the connection succeeds, a socket in the Internet domain of type SOCK_STREAM is returned to the caller and given to the remote command as **stdin** and **stdout**. If *fd2p* is nonzero, an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in the integer pointed to by *fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel and will also accept bytes on this channel representing CLIX signal numbers to be forwarded to the process group of the command. If *fd2p* is 0, the **stderr** (unit 2 of the remote command) will be the same as the **stdout** with no provision for sending arbitrary signals to the remote process, although out-of-band data could be used to attract its attention.

The protocol is described in detail in *rshd*(1M).

The *rresvport* routine is used to obtain a socket with a privileged address bound to it. This socket is suitable for use by *rcmd* and several other routines. Privileged Internet ports are in the 0 to 1023 range. Only the super-user is allowed to bind this type of address to a socket.

ruserok assumes a remote host's name returned by a *gethostbyaddr*(3B) routine, with two user names and a flag indicating whether the local user's name is also the super-user's. It then checks the files **/etc/hosts.equiv** and possibly **.rhosts** in the user's home directory to see if the request for service is allowed. A 0 is returned if the machine name is listed in the **/etc/hosts.equiv** file or the host and remote user name are in the **.rhosts** file; otherwise *ruserok* returns -1. If the *superuser* flag is 1, the checking of the **/etc/host.equiv** file is bypassed. If the local domain (as obtained from *gethostname*(2B)) is the same as the remote domain, only the machine name should be specified.

SEE ALSO

rcmd(1), rexec(3B).
rexecd(1M), rshd(1M) in the *CLIX System Administrator's Manual.*

DIAGNOSTICS

rcmd returns a valid socket descriptor on success. It returns -1 on error and prints a diagnostic message on the standard error.

rresvport returns a valid, bound socket descriptor on success. It returns -1 on error with the global value *errno* set according to the reason for failure. The error code EAGAIN is overloaded to mean "All network ports in use."

NAME

rexec - return stream to a remote command

SYNOPSIS

```
int rexec (ahost, inport, user, passwd, cmd, fd2p)
char **ahost;
int inport;
char *user, *passwd, *cmd;
int *fd2p;
```

DESCRIPTION

rexec looks up the host **ahost* using *gethostbyname*(3B), returning -1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host. If a user name and password are both specified, these are used to authenticate to the foreign host; otherwise the environment and then the user's **.netrc** file in the home directory are searched for appropriate information. If all this fails, the user is prompted for the information.

The port *inport* specifies which well-known Defense Advanced Research Project Agency (DARPA) Internet port to use for the connection; the call **getservbyname("exec", "tcp")** (see *getservent*(3B)) will return a pointer to a structure, which contains the necessary port. The protocol for connection is described in detail in *rexecd*(1M).

If the connection succeeds, a socket in the Internet domain of type SOCK_STREAM is returned to the caller and given to the remote command as **stdin** and **stdout**. If *fd2p* is nonzero, an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in the integer pointed to by *fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on the channel as being CLIX signal numbers to be forwarded to the process group of the command. The diagnostic information returned does not include remote authorization failure, as the secondary connection is set up after authorization has been verified. If *fd2p* is 0, the **stderr** (unit 2 of the remote command) will be the same as the **stdout** with no provision for sending arbitrary signals to the remote process. Although, out-of-band data could be used to attract its attention.

SEE ALSO

gethostent(3B), getservent(3B), rcmd(3B).
rexecd(1M) in the *CLIX System Administrator's Manual.*

NAME

string: index, rindex - string operations

SYNOPSIS

char *index (s, c)
char *s, c;

char *rindex (s, c)
char *s, c;

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

index (*rindex*) returns a pointer to the first (last) occurrence of character *c* in string *s*, or zero if *c* is not in the string.

3N

NAME

intro - introduction to Intergraph communications environment

DESCRIPTION

This section describes the available networking modules and routines provided with the Intergraph Network Core (INC) product. The communications library provided with the INC product contains a number of functionally separate modules. The following modules are available:

fmu	the File Management Utility (provides an interface for file transfer)
sni	the Simple Network Interface
clh	the clearinghouse

When a program is linked, the following syntax should be used:

cc *object* ... **-linc -ldevi_s**

DIAGNOSTICS

If the request is successful, a null pointer is returned. Otherwise, a pointer to an error message is returned.

NAME

clh_vbyop - lookup value by object and property

SYNOPSIS

char *clh_vbyop (object, property, value, size)
unsigned char *object, *property, *value;
int size;

DESCRIPTION

clh_vbyop searches the Intergraph clearinghouse directory looking for *object*. If found, the *object* is searched for *property*. All information (up to *size* bytes) regarding *property* is returned in the string pointed to by *value*.

FILES

/usr/lib/nodes/*	clearinghouse directory

SEE ALSO

clh(1), clh(4).

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error string is returned.

NAME

fmu_connect, fmu_disconnect - connect/disconnect to remote FMU server

SYNOPSIS

char *fmu_connect (node)
char *node;

char *fmu_disconnect ()

DESCRIPTION

fmu_connect connects with the remote File Management Utility (FMU) server on the node specified, allowing other FMU subroutines to be used.

Node must have the following format:

node-name [*.user-name* [*.password*]]

where *node-name* may be a string or network address, *user-name* and *password* are strings, and periods are the required delimiters. *User-name* and *password* may be required depending on the *node-name* specified.

fmu_disconnect ends the connection to the FMU server.

SEE ALSO

fmu(1), fmu_send(3N), fmu_receive(3N), sni_connect(3N).

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error message is returned.

WARNINGS

Connection can only be established for one node at a time. Attempting to connect to another node automatically disconnects the previous connection. If a node name is specified in *node*, the node name must be in the local Intergraph clearinghouse.

Server resources are wasted if *fmu_disconnect* is not called before exiting.

NAME

fmu_rcmd - execute the specified command on remote system

SYNOPSIS

char *fmu_rcmd (command)
char *command;

DESCRIPTION

fmu_rcmd executes *command* on the remote system. All standard output generated by the remote command is written to **stdout**.

The connection must be established with *fmu_connect*(3N) prior to calling *fmu_rcmd*.

SEE ALSO

fmu_receive(3N), fmu_send(3N), fmu_connect(3N).

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error message is returned.

WARNINGS

If *command* requires input, *fmu_rcmd* will hang indefinitely.

NAME

fmu_receive - receive files from a remote system

SYNOPSIS

char *fmu_receive (srcfile, dstfile)
char *srcfile, *dstfile;

DESCRIPTION

fmu_receive copies one or more files from a remote system to the local system.

srcfile points to a path name specifying the source file(s) on the remote system, and *dstfile* points to a path name specifying the destination file (or directory) on the local system.

The connection must be established with *fmu_connect*(3N) before *fmu_receive* is called.

SEE ALSO

fmu_send(3N), fmu_connect(3N).

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error message is returned.

NAME

fmu_send - send files to a remote system

SYNOPSIS

char *fmu_send (srcfile, dstfile)
char *srcfile, *dstfile;

DESCRIPTION

fmu_send copies one or more files from a local system to a remote system.

srcfile points to a path name specifying the source file(s) on the local system, and *dstfile* points to a path name specifying the destination file (or directory) on the remote system.

The connection must be established with *fmu_connect*(3N) before *fmu_send* is called.

SEE ALSO

fmu_receive(3N), fmu_connect(3N).

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error message is returned.

NAME

fmu_set - set FMU modes

SYNOPSIS

```
char *fmu_set (mode)
char *mode;
```

DESCRIPTION

Different modes of operation may be enabled when using the File Management Utility (FMU) routines. The following strings are supported for *mode*:

"CHECKSUM"	Force a checksum to occur during the transfer.
"COMPRESS"	Force compress mode when transferring files.
"VERBOSE"	Enable verbose mode; information about file transfer is printed to **stderr**.
"NO CHECKSUM"	Turn off checksum mode.
"NO COMPRESS"	Turn off compress mode.
"NO VERBOSE"	Turn off verbose mode.

SEE ALSO

fmu_connect(3N), fmu_receive(3N), fmu_send(3N).

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error message is returned.

NAME

rtc_allocate, rtc_deallocate, rtc_notify - remote tape control

SYNOPSIS

```
char *rtc_allocate (cdev, sys, rew, norew, timeout)
char *cdev, *sys, *rew, *norew;
int timeout;

char *rtc_deallocate (cdev)
char *cdev;

char *rtc_notify (cdev, sig)
char *cdev;
int sig;
```

DESCRIPTION

rtc_allocate, *rtc_deallocate*, and *rtc_notify* allow a tape drive on a remote machine to be used as if it resided locally (see *rtc*(7S).

rtc_allocate sets up the information needed to access the tape drive. Once allocated, the remote tape drive remains allocated until a timeout occurs or *rtc_deallocate* is executed. *Cdev* is the name of a tape control device (such as **/dev/rmt/rt0.ctl**) that controls the functions of other tape devices in the same group. *Sys* is the name or address of a remote machine with a tape drive. *Rew* and *norew* are the names of the rewind and no-rewind tape devices on the machine *sys*. *Timeout* is the number of minutes the tape can remain idle. After *timeout* idle minutes expire, a warning is sent to the system console and the tape is deallocated.

rtc_deallocate closes the connection to the remote machine to which the *cdev* control device is attached. If the tape drive is being used when the deallocate command is invoked, an error is returned.

rtc_notify sets up the signal *sig* to be sent to the calling process when *rtc_deallocate* is executed on the control device *cdev*.

Since *rtc*(7S) is a STREAMS driver, the control device that *rtc_allocate* opens must remain open to preserve the network connection. If the process that calls *rtc_allocate* needs to exit, it must *fork*(2) a child process that can use *rtc_notify* to be notified when *rtc_deallocate* is executed. Once *rtc_notify* is executed, the child process can execute *pause*(2) to wait for the notify signal.

FILES

/dev/rmt/rt?.ctl	control device
/dev/rmt/rt?	rewind *rtc*(7S) device
/dev/rmt/rt?n	no-rewind *rtc*(7S) device

SEE ALSO

rtc(1).
rtc_s(1M), rtc(7S) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error message is returned.

NAME

sni_accept - accept a connection

SYNOPSIS

```
char *sni_accept (sd)
long *sd;
```

DESCRIPTION

sni_accept accepts a connection to a requester and assigns a Simple Network Interface (SNI) descriptor to the connection. The call completes when the requester and server connect or an error occurs.

If the call is successful, the long pointed to by *sd* will be assigned an identifier for the connection, called an SNI descriptor.

FILES

/usr/ip32/inc/server.dat	server information file

SEE ALSO

sni_close(3N), sni_connect(3N), server.dat(4).
xns_listener(1M) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error message is returned.

NOTES

sni_accept should only be invoked by a server program.

NAME

sni_close - close a connection

SYNOPSIS

```
char *sni_close (sd)
long *sd;
```

DESCRIPTION

sni_close disconnects and carries out various housekeeping functions to clean up the requester/server connection. Both the server and requester must call *sni_close* to end a connection.

Sd points to the long obtained with *sni_accept*(3N) or *sni_connect*(3N). The long specifies a unique identifier assigned to a connection, called a Simple Network Interface (SNI) descriptor. Upon successful completion, *sd* points to 0.

SEE ALSO

sni_accept(3N), sni_connect(3N).

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error message is returned.

NAME

sni_connect - connect to a server program

SYNOPSIS

char *sni_connect (sd, node, sernum, server)
long *sd;
char *node, *server;
unsigned short sernum;

DESCRIPTION

sni_connect creates a requester/server connection and assigns a Simple Network Interface (SNI) descriptor to the connection. The call completes when the requester and server connect or an error occurs.

Before *sni_connect* returns, the server must invoke the *sni_accept*(3N) call.

On successful completion, the long pointed to by *sd* is assigned an identifier for the particular connection, called an SNI descriptor.

Node points to a node specifier for the remote system. Depending on the system, *node* has one of the following formats:

node-name	
node-name.user-name.	(SNI prompts for password)
node-name.user-name.passwd	
network-address	
network-address.user-name.	(SNI prompts for password)
network-address.user-name.passwd	

Sernum is the number of the server for which the connection is desired. The file **/usr/ip32/inc/server.dat** contains the available servers and server numbers for the system. If *sernum* is zero, *server* should point to the name of the executable file to be used on the remote system as a server. If *sernum* is not zero, *server* should be zero.

FILES

/usr/ip32/inc/server.dat	server information file
/usr/lib/nodes/heard/*	node name clearinghouse files

SEE ALSO

sni_accept(3N), sni_close(3N), server.dat(4).
xns_listener(1M) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error message is returned.

NOTES

sni_connect is only called in the requester program.

WARNINGS

A node name in the *node* argument may only be specified if that node name is in the local Intergraph clearinghouse.

NAME

sni_rxw - receive a data buffer

SYNOPSIS

```
char *sni_rxw (sd, buffer, len, timeout)
long *sd, timeout;
char *buffer;
int  *len;
```

DESCRIPTION

sni_rxw receives data sent by a remote node into the buffer specified by *buffer*. The size of the buffer is specified in the integer pointed to by *len*. Upon successful completion, *len* contains the number of bytes received. Completion occurs when either a complete message is received or the receiving buffer becomes full.

Sd is a pointer to a Simple Network Interface (SNI) descriptor assigned using the *sni_accept*(3N) or *sni_connect*(3N) function.

Timeout specifies the amount of time (in milliseconds) the call waits to receive data. If data is not received in the time specified, the call is unsuccessful and a pointer to an error message is returned. A *timeout* value of 0 indicates no time constraint.

SEE ALSO

sni_accept(3N), sni_close(3N), sni_connect(3N), sni_txw(3N).

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error message is returned.

NOTES

In general, timeouts should not be used.

WARNINGS

If received data is larger than the specified size of the buffer, the data will be truncated. The remaining data will be lost.

NAME

sni_txw - transmit a data buffer

SYNOPSIS

```
char *sni_txw (sd, buffer, len, timeout)
long *sd, timeout;
char *buffer;
int *len;
```

DESCRIPTION

sni_txw transmits data to a remote node from *buffer*. The size of the buffer is specified in the integer pointed to by *len*. Upon successful completion, *len* contains the actual number of bytes transmitted.

Sd is a pointer to a Simple Network Interface (SNI) descriptor assigned using the *sni_accept*(3N) or *sni_connect*(3N) function.

Timeout specifies the amount of time (in milliseconds) the call waits for an acknowledgment from the remote node. If the acknowledgment is not received in the time specified, the call is unsuccessful and a pointer to an error message is returned. A *timeout* value of 0 indicates no time constraint.

SEE ALSO

sni_accept(3N), sni_close(3N), sni_connect(3N), sni_rxw(3N).

DIAGNOSTICS

Upon successful completion, a null pointer is returned. Otherwise, a pointer to an error message is returned.

NOTES

In general, timeouts should not be used.

WARNINGS

sni_txw completes when the *buffer* is queued. The data is transmitted asynchronously until all the is successfully received by the remote node.

3R

NAME

intro - introduction to RPC/XDR/YP service functions and protocols

DESCRIPTION

These functions define access routines to the standard Remote Procedure Call (RPC) and Yellow Pages (YP) services. To access these routines link with **libyp.a** for YP services and **libbsd.a** for RPC information.

List Of Standard Services

Name	Appears on Page	Description
getdomain	getdomain(3R)	get YP domain name
getrpcent	getrpcent(3R)	get RPC entry name
getrpcport	getrpcport(3R)	get RPC port number
ypclnt	ypclnt(3R)	YP protocol
yppasswd	yppasswd(3R)	update users passwd in YP

SEE ALSO

"RPC/XDR Tutorial", "YP Tutorial" in the *CLIX System Guide*.

NAME

getdomainname, setdomainname - get/set name of current domain

SYNOPSIS

```
int getdomainname (name, namelen)
char *name;
int namelen;

int setdomainname (name, namelen)
char *name;
int namelen;
```

DESCRIPTION

getdomainname returns the name of the domain for the host machine as previously set by *setdomainname*. The parameter *namelen* specifies the size of the *name* array. The returned name is null-terminated unless insufficient space is provided.

setdomainname sets the domain of the host machine to be *name*, which has length *namelen*. This call is restricted to the super-user and is normally used only at boot time.

Domains enable two distinct networks that may have host names in common to merge. Each network is distinguished by having a different domain name. Currently, only the Yellow Pages (YP) service uses domains.

The following errors may be returned by these calls:

[EFAULT]	The *name* parameter gave an invalid address.
[EPERM]	The caller was not the super-user. This error only applies to *setdomainname*.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NOTES

Only the super-user can set the domain name.

BUGS

Domain names are limited to 64 characters.

NAME

getrpcent, getrpcbyname, getrpcbynumber, setrpcent, endrpcent - get RPC entry

SYNOPSIS

#include <netdb.h>

struct rpcent *getrpcent ()

struct rpcent *getrpcbyname (name)
char *name;

struct rpcent *getrpcbynumber (number)
int number;

void setrpcent (stayopen)
int stayopen;

void endrpcent ()

DESCRIPTION

getrpcent, *getrpcbyname*, and *getrpcbynumber* return a pointer to an object with the following structure containing the broken-out fields of a line in the Remote Procedure Call (RPC) program number database, **/etc/rpc**.

```
struct rpcent {
        char  *r_name;      /* name of server for this rpc program */
        char  **r_aliases;  /* alias list */
        long  r_number;     /* rpc program number */
};
```

The members of this structure are as follows:

r_name	The name of the server for this RPC program.
r_aliases	A zero-terminated list of alternate names for the RPC program.
r_number	The RPC program number for this service.

getrpcent reads the next line of the file, opening the file if necessary.

setrpcent opens and rewinds the file. If the *stayopen* flag is nonzero, the net database is not closed after each call to *getrpcent* (either directly or indirectly)

endrpcent closes the file by either *getrpcbyname* or *getrpcbynumber*.

getrpcbyname and *getrpcbynumber* sequentially search from the beginning of the file until a matching RPC program name or program number is found, or until EOF is encountered.

FILES

/etc/rpc
/etc/yp/*domainname*/rpc.bynumber

SEE ALSO

rpc(4).
rpcinfo(1M) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

A null pointer is returned on EOF or error.

BUGS

All information is contained in a static area, so it must be copied to be saved.

NAME

getrpcport - get RPC port number

SYNOPSIS

```
int getrpcport (host, prognum, versnum, proto)
char *host;
int prognum, versnum, proto;
```

DESCRIPTION

getrpcport returns the port number for version *versnum* of the Remote Procedure Call (RPC) program *prognum* running on *host* and using protocol *proto*.

DIAGNOSTICS

getrpcport returns 0 if it cannot contact the portmapper or if *prognum* is not registered. If *prognum* is registered but not with version *versnum*, the portmapper returns the port number.

NAME

ypclnt: yp_get_default_domain, yp_bind, yp_unbind, yp_match, yp_first, yp_next, yp_all, yp_order, yp_master, yperr_string, ypprot_err - YP client interface

SYNOPSIS

```
#include <rpcsvc/ypclnt.h>

int yp_bind (indomain)
char *indomain;

void yp_unbind (indomain)
char *indomain;

int yp_get_default_domain (outdomain)
char **outdomain;

int yp_match (indomain, inmap, inkey, inkeylen, outval, outvallen)
char *indomain, *inmap, *inkey, **outval;
int inkeylen, *outvallen;

int yp_first (indomain, inmap, outkey, outkeylen, outval, outvallen)
char *indomain, *inmap, **outkey, **outval;
int *outkeylen, *outvallen;

int yp_next (indomain, inmap, inkey, inkeylen, outkey, outkeylen,
        outval, outvallen)
char *indomain, *inmap, *inkey, **outkey, **outval;
int inkeylen, *outkeylen, *outvallen;

int yp_all (indomain, inmap, incallback)
char *indomain, *inmap;
struct ypall_callback incallback;

int yp_order (indomain, inmap, outorder)
char *indomain, *inmap;
int *outorder;

int yp_master (indomain, inmap, outname)
char *indomain, *inmap, **outname;

char *yperr_string (incode)
int incode;

int ypprot_err (incode)
unsigned int incode;
```

DESCRIPTION

This package of functions provides an interface to the Yellow Pages (YP) network lookup service. The package can be loaded from the standard library, **/usr/lib/libyp.a**. Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of YP, including the definitions of *map* and *domain* and a description of the various servers, databases, and commands.

All input parameter names begin with "in". Output parameters begin with "out". Output parameters which are pointers to character pointers should be addresses of uninitialized character pointers. Memory is allocated by the YP client package using *malloc*(3) and may be freed if the user code has no continuing need for it. For each *outkey* and *outval*, two extra bytes of memory are allocated at the end that contain NEWLINE and NULL, respectively, but these two bytes are not reflected in *outkeylen* or *outvallen*. *Indomain* and *inmap* strings must be non-null and null-terminated. String parameters accompanied by a count parameter may not be null but may point to null strings, with the count parameter indicating this. Counted strings need not be null-terminated.

All functions that return integers return 0 if they succeed and a failure code (YPERR_*xxxx*) otherwise. Failure codes are described under **DIAGNOSTICS** below.

The YP lookup calls require a map name and a domain name, at minimum. It is assumed that the client process knows the name of the map of interest. Client processes should fetch the node's default domain by calling *yp_get_default_domain* and use the returned *outdomain* as the *indomain* parameter to successive YP calls.

To use the YP services, the client process must be "bound" to a YP server that serves the appropriate domain using *yp_bind*. Binding need not be explicitly set by user code; it is accomplished automatically when a YP lookup function is called. *yp_bind* can be called directly for processes that use a backup strategy (e.g., a local file) in cases when YP services are not available.

Each binding allocates (uses up) one client process socket descriptor; each bound domain costs one socket descriptor. However, multiple requests to the same domain use that same descriptor. *yp_unbind* is available at the client interface for processes that explicitly manage their socket descriptors while accessing multiple domains. The call to *yp_unbind* makes the domain unbound, and free all per-process and per-node resources used to bind it.

If a Remote Procedure Call (RPC) failure results when a binding is used, that domain will be unbound automatically. At that point, the ypclnt layer retries indefinitely or until the operation succeeds, provided that *ypbind*(1M) is running, and either

a) the client process cannot bind a server for the proper domain, or

b) RPC requests to the server fail.

If an error is not RPC-related, if *ypbind*(1M) is not running, or if a bound *ypserv*(1M) process returns any answer (success or failure), the ypclnt layer will return control to the user code, with either an error code or a success code and any results.

yp_match returns the value associated with a passed key. This key must be exact; no pattern matching is available.

yp_first returns the first key-value pair from the named map in the named domain.

yp_next returns the next key-value pair in a named map. The *inkey* parameter should be the *outkey* returned from an initial call to *yp_first* (to get the second key-value pair) or the one returned from the nth call to *yp_next* (to get the nth + second key-value pair).

The concept of first (and, for that matter, of next) is particular to the structure of the YP map being processed, the lexical order within any original (non-YP) database or any obvious numerical sorting order on the keys, values, or key-value pairs do not relate to retrieval order. The only ordering guarantee is that if the *yp_first* function is called on a particular map, and then the *yp_next* function is repeatedly called on the same map at the same server until the call fails with a reason of YPERR_NOMORE, every entry in the database will be seen exactly once. Further, if the same sequence of operations is performed on the same map at the same server, the entries will be seen in the same order.

When heavy server load or server failure occurs, it is possible for the domain to become unbound, then bound once again (perhaps to a different server) while a client is running. This can cause a break in one of the enumeration rules; specific entries may be seen twice by the client or not at all. This approach protects the client from error messages that would otherwise be returned in the midst of the enumeration. The next paragraph describes a better solution to enumerating all entries in a map.

yp_all provides a way to transfer an entire map from server to client in a single request using Transmission Control Protocol (TCP) (rather than User Datagram Protocol (UDP) as with other functions in this package). The entire transaction occurs as a single RPC request and response. *yp_all* can be used like any other YP procedure by identifying the map in the normal manner and supplying the name of a function that will be called to process each key-value pair within the map. A return from the call to *yp_all* only occurs when the transaction is completed (successfully or unsuccessfully) or when the *foreach* function does not want to see any more key-value pairs.

The third parameter to *yp_all* is

```
struct ypall_callback *incallback {
        int     (*foreach)( );
        char    *data;
};
```

The function *foreach* is called as follows:

```
foreach (instatus, inkey, inkeylen, inval, invallen, indata)
char *inkey, *inval, *indata;
int instatus, inkeylen, invallen;
```

The *instatus* parameter will hold one of the return status values defined in **<rpcsvc/yp_prot.h>** — either YP_TRUE or an error code. (See *ypprot_err*, below, for a function that converts a YP protocol error code to a ypclnt layer error code.)

The key and value parameters differ somewhat from those defined in the synopsis section above. First, the memory pointed to by the *inkey* and *inval* parameters is private to the *yp_all* function and is overwritten with the arrival of each new key-value pair. It is the responsibility of the *foreach* function to use the contents of that memory, but it does not own the memory itself. Key and value objects presented to the *foreach* function look exactly as they do in the server's map; if they were not newline-terminated or null-terminated in the map, they will not be so here either.

The *indata* parameter is the contents of the *incallback data* element passed to *yp_all*. The *data* element of the *callback* structure may be used to share state information between the *foreach* function and the mainline code. Its use is optional and no part of the YP client package inspects its contents.

The *foreach* function returns zero to indicate that it wants to be called again for further received key-value pairs, or nonzero to stop the flow of key-value pairs. If *foreach* returns a nonzero value, it is not called again; the functional value of *yp_all* is then 0.

yp_order returns the order number for a map.

yp_master returns the machine name of the master YP server for a map.

yperr_string returns a pointer to an error message string that is null-terminated but contains no period or new line.

ypprot_err has a YP protocol error code as input and returns a ypclnt layer error code, which may be used as input to *yperr_string*.

FILES

/usr/include/rpcsvc/ypclnt.h
/usr/include/rpcsvc/yp_prot.h

SEE ALSO

ypfiles(4).
ypserv(1M) in the *CLIX System Administrator's Reference Manual.*
"YP Tutorial" in the *CLIX System Guide.*

DIAGNOSTICS

All integer functions return 0 if the requested operation is successful or one of the following errors if the operation fails.

```
#define YPERR_BADARGS    1  /* args to function are bad */
#define YPERR_RPC        2  /* RPC failure - domain unbound */
#define YPERR_DOMAIN     3  /* can't bind to server on this domain */
#define YPERR_MAP        4  /* no such map in server's domain */
#define YPERR_KEY        5  /* no such key in map */
#define YPERR_YPERR      6  /* internal yp server or client error */
#define YPERR_RESRC      7  /* resource allocation failure */
#define YPERR_NOMORE     8  /* no more records in map database */
#define YPERR_PMAP       9  /* can't communicate with portmapper */
#define YPERR_YPBIND     10 /* can't communicate with ypbind */
```

```
#define YPERR_YPSERV        11  /* can't communicate with ypserv */
#define YPERR_NODOM         12  /* local domain name not set */
```

NAME

yppasswd - update user password in YP

SYNOPSIS

```
#include <rpcsvc/yppasswd.h>

int yppasswd (oldpass, newpw)
char *oldpass;
struct passwd *newpw;
```

DESCRIPTION

If *oldpass* is a valid user password, *yppasswd* replaces *oldpass* with *newpw*.

Remote Procedure Call info:

```
program number:
        YPPASSWDPROG

xdr routines:
        xdr_ppasswd(xdrs, yp)
                XDR *xdrs;
                struct yppasswd *yp;
        xdr_yppasswd(xdrs, pw)
                XDR *xdrs;
                struct passwd *pw;
procs:
        YPPASSWDPROC_UPDATE
                Takes yppasswd structure as argument; returns integer.
                Same behavior as yppasswd wrapper.
                Uses UNIX authentication.
versions:
        YPPASSWDVERS_ORIG

structures:
        struct yppasswd {
                char *oldpass;          /* old (unencrypted) password */
                struct passwd newpw; /* new pw structure */
        };
```

SEE ALSO

yppasswd(1).
yppasswdd(1M) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned.

3A

NAME

intro - introduction to the synchronous/asynchronous I/O library

DESCRIPTION

This section describes the routines found in the synchronous/asynchronous I/O (XIO) library, **libix**. The link editor *ld*(1) searches this library under the **-lix** option.

The XIO system controls synchronous and asynchronous I/O requests to XIO drivers in the CLIX kernel (see section (7A) in the *CLIX System Administrator's Reference Manual*). Although any request to an XIO driver may be asynchronous, only requests that could block are available as asynchronous requests.

Associated with each asynchronous request is an event flag number (*efn*) and an XIO completion status block (*xiosb* structure) passed as part of the parameter list. The event flag number is used to track the completion of the asynchronous request; the *xiosb* structure is updated with the completion information for the asynchronous request.

Two functions in the library manage allocation of event flag numbers. The function *xio_allocef*(3A) allocates an unused event flag number (an integer between 0 and 31, inclusive). The *xio_deallocef*(3A) function frees a previously allocated event flag number.

The *xiosb* structure, allocated for each asynchronous request, is shown below and is defined in the include file **<sys/xio/xio.h>**.

```
struct xiosb {
        int     status;     /* completion error status */
        int     xfcnt;      /* size of transferred data */
        int     seqnum;     /* sequence number of event */
};
```

The members of the *xiosb* structure are described as follows:

status The error value returned from the XIO driver. Possible return values are listed in the **<sys/xio/xerr.h>** include file. A value of 0 indicates no error occurred.

xfcnt The total number of bytes of data transferred by the XIO driver. However, in some cases, a parameter may be returned in this member. See the manual page for each request to determine the exact use of this member.

seqnum This member determines the order of completion for asynchronous requests. The XIO system sets this member to a value of one greater than the previously completed asynchronous request.

The XIO system in the CLIX kernel provides a 32-bit mask, the event flag mask, to monitor the completion of asynchronous requests. Initially, all bits in the event flag mask are set. When an asynchronous request is issued, the XIO system clears the bit corresponding to the event flag number for the request. When an asynchronous request has completed, the XIO system sets the bit corresponding to the event flag number of the completed request. The user may determine the completion of an asynchronous request by checking its corresponding bit in the event flag mask.

In addition to bits in the event flag mask being altered by the invocation and completion of asynchronous requests, the following routines also modify the event flag mask. *xio_clref*(3A) clears bits in the event flag mask as specified in the calling argument, and *xio_setef*(3A) sets bits in the event flag mask as specified in the calling argument.

Four routines allow a process to act on the current state of the event flag mask (and thus the completion state of the associated asynchronous requests). The routines are described as follows:

xio_readef(3A)	Return the event flag mask. Cleared bits in the mask possibly represent asynchronous requests that have not completed. The process is responsible for knowing which bits in the event flag mask are currently used by an asynchronous request.
xio_waitfr(3A)	Return control to the caller only when the bit in the event flag mask corresponding to the event flag number specified in the call is set. The call, in effect, waits for the completion of an outstanding asynchronous request.
xio_wflor(3A)	Return control to the caller when any of the bits in the event flag mask that correspond to set bits in the mask specified by the call are set. The call, in effect, waits for one of the outstanding asynchronous requests to complete.
xio_wfland(3A)	Return control to the caller when all bits in the event flag mask that correspond to all set bits in the mask specified by the call are set. This provides a mechanism for a process to wait for the completion of many outstanding asynchronous requests.

In addition to the above routines, *xio_notify*(3A) notifies a process that an asynchronous request has been completed. The notification is either by a signal, by alternation of an integer pair in the process's data space, or both. See *xio_notify*(3A) for more information.

EXAMPLES

The following example uses an asynchronous request to receive data from a device. The device DEV is fictitious and used with this example only.

```
#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>
#include <errno.h>

...

        int     efn;
        struct xiosb xiosb;
        char    buffer[SIZE];
        ...
        /*
         * Allocate unique event number
         */
        if (xio_allocef(&efn) == 0) {
                printf("no more events\n");
                exit(1);
        }
        /*
         * Issue request to device DEV
         */
        if (DEV_readnw(buffer, SIZE, &xiosb, efn) == XIO_FAILURE) {
                printf("request failed, errno = %d\n", errno);
                exit(1);
        }
        /*
         * At this point, the process is free to do any other type
         * of requests or processing.  When the "DEV_readnw" event
         * completes, the request will be queued until the process
         * polls the system or waits for this event.
         *
         * For the purpose of our example, the program will wait for
         * this event.
         */
        xio_waitfr(efn);
        if (xiosb.status) {
                printf("bad request status, status = %d\n", xiosb.status);
                exit(1);
        }
        printf("total amount of data returned = %d\n", xiosb.xfcnt);

        ...
```

SEE ALSO

xio_allocef(3A), xio_readef(3A), xio_notify(3A), xio_waitfr(3A).
Section (7A) in the *CLIX System Administrator's Reference Manual.*

WARNINGS

If any Environ V Library functions for graphics or graphics device interfaces are used, the XIO system is controlled by Environ V functions. To use XIO devices with Environ V functions, refer to *Setup_system_event*(3C) in the *Environ V Programmer's Reference Manual.*

Because an event flag number is used to track an asynchronous request completion, no two outstanding requests should use the same event flag number at the same time.

CAVEATS

Since XIO devices are not in the CLIX name space, access to them cannot be restricted.

NAME

aux_break - generate a break on a serial port

SYNOPSIS

#include <sys/xio/xerr.h>

int aux_break (port)
int port;

DESCRIPTION

aux_break generates a break condition on the specified serial *port*. The *port* must have been opened by *aux_open*(3A).

aux_break will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[AUX_PORT_NOT_OPEN]	The specified *port* is not open by the calling process.

SEE ALSO

aux_open(3A).
xaux(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

aux_cancel - cancel outstanding read on a serial port

SYNOPSIS

#include <sys/xio/xerr.h>

```
int aux_cancel (port)
int port;
```

DESCRIPTION

aux_cancel cancels the outstanding read on the specified serial *port*. The *port* must have been opened by *aux_open*(3A).

aux_cancel will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[AUX_PORT_NOT_OPEN]	The specified *port* is not open by the calling process.

SEE ALSO

aux_open(3A), aux_read(3A), aux_rawrd(3A).
xaux(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

CAVEATS

Outstanding writes cannot be canceled.

NAME

aux_cancel_modem - cancel modem change state on a serial port

SYNOPSIS

#include <sys/xio/xerr.h>

int aux_cancel_modem (port)
int port;

DESCRIPTION

aux_cancel_modem cancels the outstanding request for modem state change on the specified serial *port*. The *port* must have been opened by *aux_open*(3A).

aux_cancel_modem will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[AUX_PORT_NOT_OPEN]	The specified *port* is not open by the calling process.

SEE ALSO

aux_open(3A), aux_modem(3A).
xaux(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

aux_close - close a serial port

SYNOPSIS

```
#include <sys/xio/xerr.h>

int aux_close (port)
int port;
```

DESCRIPTION

aux_close cancels outstanding I/O on the specified serial *port* and closes the *port*. The *port* must have been opened by *aux_open*(3A).

aux_close will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[AUX_PORT_NOT_OPEN]	The specified *port* is not open by the calling process.

SEE ALSO

aux_open(3A), aux_read(3A), aux_rawrd(3A), aux_modem(3A).
xaux(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

CAVEATS

Outstanding writes cannot be canceled.

NAME

aux_modem, aux_modem_nw - get modem change from a serial port

SYNOPSIS

```
#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>

int aux_modem (port)
int port;

int aux_modem_nw (port, xiosb, efn)
int port, efn;
struct xiosb *xiosb;
```

DESCRIPTION

aux_modem waits for a change of modem state on the specified *port*. The *port* must have been opened by *aux_open*(3A).

To get and set modem characteristics, refer to *aux*(7S).

aux_modem_nw is the asynchronous version of *aux_modem*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated on completion of the request (see *intro*(3A)).

aux_modem and *aux_modem_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[AUX_PORT_NOT_OPEN]	The specified *port* is not open by the calling process.

SEE ALSO

intro(3A), aux_open(3A).
xaux(7A), aux(7S) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

aux_open - open a serial port

SYNOPSIS

```
#include <sys/xio/xerr.h>

int aux_open (port)
int port;
```

DESCRIPTION

aux_open opens the specified serial *port* if it is not currently opened by either this call or by the standard *open*(2) call.

aux_open will default the modem characteristics of the *port* to the standard *open*(2) state. The modem characteristics may be changed after a successful *aux_open* by opening the corresponding **/dev/tty??** device, and issuing the appropriate *ioctl*(2).

aux_open will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[AUX_PORT_IN_USE]	The specified *port* is already in use.

SEE ALSO

aux_close(3A).
xaux(7A), aux(7S) in the *CLIX System Administrator's Reference Manual.*
open(2), ioctl(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

aux_rawrd, aux_rawrd_nw - read data with error byte from a serial port

SYNOPSIS

```
#include <sys/aux.h>
#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>

int aux_rawrd (port, dbuf, dcnt, xfcnt)
int port, dcnt;
short *dbuf;
int *xfcnt;

int aux_rawrd_nw (port, dbuf, dcnt, xiosb, efn)
int port, dcnt, efn;
short *dbuf;
struct xiosb *xiosb;
```

DESCRIPTION

aux_rawrd reads data and error information from the specified *port*. The *port* must have been opened by *aux_open*(3A).

Each short read from *port* consists of a character in the low byte and error information in the high-order byte. The error information is a bit mask with bit definitions shown below and defined in **<sys/aux.h>**. Multiple errors can occur on a character.

BREAK	0x0100	/* break sequence detected */
PARITY	0x0200	/* character parity error */
PART_OVERRUN	0x0400	/* hardware overrun */
FRAME	0x0800	/* character framing error */
QUEUE_OVERRUN	0x1000	/* software overrun */

Dbuf is a pointer to the buffer to receive the character/error information, and *dcnt* is the size (in bytes) of the buffer. The buffer must be short aligned.

Upon completion of the synchronous request, the integer pointed to by *xfcnt* is updated with the number of bytes transferred to *dbuf*. Since each character received from *port* is coupled with error information, *xfcnt* indicates twice as many bytes as there were characters received.

aux_rawrd_nw is the asynchronous version of *aux_rawrd*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred to *dbuf*.

aux_rawrd and *aux_rawrd_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[AUX_PORT_NOT_OPEN]	The specified *port* is not open by the calling process.
[AUX_PORT_REDUNDANT_REQUEST]	An outstanding read is already on the specified *port*.
[BAD_DATA_BUFFER_SIZE]	*Dcnt* is less than or equal to zero.
[BAD_DATA_BUFFER_ADDRESS]	*Dbuf* points to a nonwritable memory space.

SEE ALSO

intro(3A), aux_open(3A).
xaux(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

aux_read, aux_read_nw - read data from a serial port

SYNOPSIS

```
#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>

int aux_read (port, dbuf, dcnt, xfcnt)
int port, dcnt;
char *dbuf;
int *xfcnt;

int aux_read_nw (port, dbuf, dcnt, xiosb, efn)
int port, dcnt, efn;
short *dbuf;
struct xiosb *xiosb;
```

DESCRIPTION

aux_read reads data from the specified *port*. The *port* must have been opened by *xaux_open*(3A).

Dbuf is a pointer to the buffer to receive the characters, and *dcnt* is the size (in bytes) of the buffer.

Upon completion of the synchronous request, the integer pointed to by *xfcnt* is updated with the number of bytes transferred to *dbuf*.

aux_read_nw is the asynchronous version of *aux_read*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred to *dbuf*.

aux_read and *aux_read_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[AUX_PORT_NOT_OPEN]	The specified *port* is not open by the calling process.
[AUX_PORT_REDUNDANT_REQUEST]	An outstanding read is already on the specified *port*.
[BAD_DATA_BUFFER_SIZE]	*Dcnt* is less than or equal to zero.
[BAD_DATA_BUFFER_ADDRESS]	*Dbuf* points to a nonwritable memory space.

SEE ALSO

intro(3A), aux_open(3A).
xaux(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

aux_write, aux_write_nw - write data to a serial port

SYNOPSIS

```
#include <sys/xio/xaux.h>
#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>

int aux_write (port, dbuf, dcnt, xfcnt)
int port, dcnt;
char *dbuf;
int *xfcnt;

int aux_write_nw (port, dbuf, dcnt, xiosb, efn)
int port, dcnt, efn;
short *dbuf;
struct xiosb *xiosb;
```

DESCRIPTION

aux_write writes data to the specified *port*. The *port* must have been opened by *xaux_open*(3A).

Dbuf is a pointer to a buffer of characters to be written, and *dcnt* is the size (in bytes) of the buffer.

Upon completion of the synchronous request, the integer pointed to by *xfcnt* is updated with the number of bytes transmitted.

aux_write_nw is the asynchronous version of *aux_write*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transmitted.

aux_write and *aux_write_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[AUX_PORT_NOT_OPEN]	The specified *port* is not open by the calling process.
[BAD_DATA_BUFFER_SIZE]	*Dcnt* is less than or equal to zero or greater than NAUXWRITE.
[BAD_DATA_BUFFER_ADDRESS]	*Dbuf* points to an invalid memory space.

SEE ALSO

intro(3A), aux_open(3A).
xaux(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

cnv_close - close a CNV channel

SYNOPSIS

#include <sys/xio/xerr.h>

int cnv_close (channel)
int channel;

DESCRIPTION

cnv_close unmaps any hardware registers associated with the specified Convolution Filter (CNV) *channel* and closes the *channel*. *Channel* must have been opened with *cnv_open*(3A).

cnv_close(3A) will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[CNV_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[CNV_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

cnv_open(3A).
xcnv(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

cnv_open - open a CNV channel

SYNOPSIS

#include <sys/xio/xerr.h>

int cnv_open (channel, base)
int channel, *base;

DESCRIPTION

cnv_open opens the specified Convolution Filter (CNV) *channel.* Only one process can open a CNV channel at a time. Each CNV board in the system is represented by a *channel* number so that *channel* 0 references the CNV board with the lowest Shared Resource (SR) Bus slot number for all CNVs. If the call is successful, *base* will contain the virtual base address of the specified CNV board. All convolution parameters are available to the calling process through this mapping.

cnv_open will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[CNV_REDUNDANT_REQ]	The specified *channel* is currently opened by this process.
[CNV_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[CNV_CHANNEL_NOT_FOUND]	The specified *channel* is not in the system.
[CNV_CHANNEL_BUSY]	The specified *channel* is currently opened by another process.

SEE ALSO

xcnv(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

csi_cancel - cancel outstanding asynchronous I/O on a CSI port

SYNOPSIS

#include <sys/xio/xerr.h>

int csi_cancel (channel)
int channel;

DESCRIPTION

csi_cancel terminates all pending commands and death events associated with the calling process on the Command Status Interface (CSI) port of the Image System Interface (ISI) board referenced by *channel*. *Channel* must have been opened with *csi_open*(3A).

csi_cancel will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

csi_open(3A), csi_cmd(3A), csi_dstat(3A), csi_death(3A).
xcsi(7A) in the *CLIX System Administrator's Reference Manual*.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

csi_ccan - cancel a specific command on a CSI port

SYNOPSIS

#include <sys/xio/xerr.h>

int csi_ccan (channel, cmd)
int channel, cmd;

DESCRIPTION

csi_ccan terminates the pending *cmd* associated with the calling process on the Command Status Interface (CSI) port of the Image System Interface (ISI) board referenced by *channel*. *Channel* must have been opened with *csi_open*(3A).

csi_ccan will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

csi_open(3A), csi_cmd(3A), csi_dstat(3A).
xcsi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

csi_close - close a CSI port

SYNOPSIS

```
#include <sys/xio/xerr.h>

int csi_close (channel)
int channel;
```

DESCRIPTION

csi_close terminates all pending requests associated with the calling process on the Command Status Interface (CSI) port of the Image System Interface (ISI) board referenced by *channel* and closes the port. *Channel* must have been opened with *csi_open*(3A).

csi_close will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

csi_open(3A), csi_cmd(3A), csi_dstat(3A), csi_ustat(3A), csi_death(3A).
xcsi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

csi_cmd, csi_cmd_nw - send command packets to a CSI port

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int csi_cmd (channel, cbuf, ccnt, sbuf, scnt, timeout, xfcnt)
int channel, ccnt, scnt, timeout;
char *cbuf, *sbuf;
int *xfcnt;

int csi_cmd_nw (channel, cbuf, ccnt, sbuf, scnt, timeout, xiosb, efn)
int channel, ccnt, scnt, timeout, efn;
char *cbuf, *sbuf;
struct xiosb *xiosb;
```

DESCRIPTION

csi_cmd provides a mechanism to send a command packet and receive the associated status packet over the Command Status Interface (CSI) port on the Image System Interface (ISI) board referenced by *channel*. *Channel* must have been opened with *csi_open*(3A).

Before sending the command packet pointed to by *cbuf*, the packet header is copied into the kernel and the source ID, magic numbers, start time, and checksum are filled in. Bit 15 of the command word is never asserted on outgoing command headers, and bit 15 of the status word is always ignored on incoming status headers. *Cbuf* must point to a long-word aligned buffer.

Ccnt is the size (in bytes) of the command packet. It must be greater than or equal to twice the word count submitted in the command packet header. The packet header size is 10 words, hence, the minimum value for *ccnt* is 20.

The status packet received from the CSI port for the command is copied into the buffer pointed to by *sbuf*. The fields are checked and the finish time is entered in the appropriate field of the status packet header. *Sbuf* must point to a long-word aligned buffer.

Scnt is the size (in bytes) of *sbuf*. If *scnt* is 0, then the request will complete as soon as the command packet is sent. Otherwise, *scnt* must also be greater than or equal to 20.

Timeout is the time limit in 1/60 second intervals to receive a status packet after sending the command packet. Any transaction that takes longer is aborted and an appropriate status is returned. A *timeout* value of 0 disables the timeout function.

Upon completion of the synchronous request, the integer pointed to by *xfcnt* is updated with the number of bytes received in the status packet.

csi_cmd_nw is the asynchronous version of *csi_cmd*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes received in the status packet.

csi_cmd and *csi_cmd_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[BAD_DATA_BUFFER_ADDRESS]	The command and status packets are not long-word aligned or lack the necessary permission.
[BAD_DATA_BUFFER_SIZE]	A nonzero packet size is smaller than the header size of 20 bytes.
[PAGE_LOCK_FAILED]	Not enough physical memory for this request is available.
[ISI_CANCELED]	The request was canceled by either *csi_close*(3A), *csi_ccan*(3A), *csi_reset*(3A), or *csi_cancel*(3A).
[ISI_PARITY_ERROR]	A parity error occurred during the transaction.
[ISI_CYCLE_ERROR]	A hardware handshake error occurred during the transaction.
[ISI_TIMEOUT]	The *timeout* expired before the transaction completed.
[ISI_PROTOCOL_ERROR]	A firmware handshake error occurred during the transaction. This probably means that *scnt* was smaller than the actual size of the status packet.
[ISI_HARDWARE_CHECK]	A fatal status code was asserted on the DR11 status lines during the transaction.

SEE ALSO

intro(3A), csi_open(3A), csi_cancel(3A), csi_ccan(3A), csi_reset(3A).
xcsi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

csi_death, csi_death_nw - wait for a CSI communication to fail

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int csi_death (channel, timeout, reason)
int channel, timeout;
int *reason;

int csi_death_nw (channel, timeout, xiosb, efn)
int channel, timeout, efn;
struct xiosb *xiosb;
```

DESCRIPTION

csi_death provides a mechanism to sense that communication through the Command Status Interface (CSI) port on the Image System Interface (ISI) board referenced by *channel* has failed. *Channel* must have been opened with *csi_open*(3A).

Timeout is the time limit in 1/60 second intervals to sense a failure. Upon expiration, the request is aborted and an appropriate status is returned. A value of 0 disables the timeout function.

Upon completion of the synchronous request, the integer pointed to by *reason* will indicate why the failure occurred. Possible reasons for failure are application specific and are copied from the DR11 status lines which connect directly to external hardware. DR11 status codes 4 through 6 are reserved as fatal status codes. A status code of 7 indicates a cable problem.

csi_death_nw is the asynchronous version of *csi_death*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the reason why the failure occurred.

csi_death and *csi_death_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[ISI_CANCELED]	The request was canceled by either *csi_cancel*(3A) or *csi_close*(3A).
[ISI_TIMEOUT]	The *timeout* expired before a failure occurred.

SEE ALSO

intro(3a), csi_open(3A), csi_close(3A), csi_cancel(3A).
xcsi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

csi_dstat_nw - receive delayed status from a CSI port

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int csi_dstat_nw (channel, cmd, sbuf, scnt, timeout, xiosb, efn)
int channel, cmd, scnt, timeout, efn;
char *sbuf;
struct xiosb *xiosb;
```

DESCRIPTION

csi_dstat_nw provides a mechanism to establish a receive buffer to catch delayed status over the Command Status Interface (CSI) port on the Image System Interface (ISI) board referenced by *channel*. *Channel* must have been opened with *csi_open*.

There is no synchronous version of this function since it is necessary to start the *csi_cmd*(3A) after establishing delayed status. Any other order of operation would introduce timing windows.

Cmd is the command identifier to associate the delayed status packet with a specific command packet to be sent at a later time.

Sbuf is a pointer to the delayed status buffer and must begin on a long-word boundary. *Scnt* is the size (in bytes) of *sbuf*. This value must be greater than or equal to 20 which is the packet header size.

Timeout is the time limit in 1/60 second intervals to receive the delayed status packet. Upon expiration, the request is aborted and an appropriate status is returned. A *timeout* value of 0 disables the timeout function.

Efn is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred to the status buffer.

csi_dstat_nw will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[BAD_DATA_BUFFER_ADDRESS]	*Sbuf* is not long-word aligned or points to a nonwritable memory space.
[BAD_DATA_BUFFER_SIZE]	The *scnt* is smaller than the header size of 20 bytes.
[PAGE_LOCK_FAILED]	There is not enough physical memory available for this request at this time.

[ISI_CANCELED]	The request was canceled with either *csi_close*(3A), *csi_ccan*(3A), *csi_reset*(3A), or *csi_cancel*(3A).
[ISI_PARITY_ERROR]	A parity error occurred during the transfer.
[ISI_CYCLE_ERROR]	A hardware handshake error occurred during the transfer.
[ISI_TIMEOUT]	The *timeout* expired before the status packet was received.
[ISI_PROTOCOL_ERROR]	A firmware handshake error occurred during the transfer. This probably means that *scnt* was smaller than the actual size of the status packet.
[ISI_HARDWARE_CHECK]	A fatal status code was asserted on the DR11 status lines during the transfer.

SEE ALSO

intro(3A), csi_open(3A), csi_close(3A), csi_cancel(3A), csi_ccan(3A), csi_reset(3A).
xcsi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

csi_open - open a CSI port

SYNOPSIS

```
#include <sys/xio/xerr.h>

int csi_open (channel)
int channel;
```

DESCRIPTION

csi_open opens the Command Status Interface (CSI) port on the Image System Interface (ISI) board referenced by *channel.* Multiple processes can have the same CSI port open at a time. Each ISI board in the system is represented by a *channel* number such that channel 0 references the ISI board with the lowest Shared Resource Bus slot number of all ISI boards.

csi_open will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[ISI_REDUNDANT_REQ]	The specified *channel* is currently open by this process.
[ISI_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[ISI_CHANNEL_NOT_FOUND]	The specified *channel* is not present in the system.

SEE ALSO

csi_close(3A).
xcsi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

csi_reset - reset hardware on CSI port

SYNOPSIS

#include <sys/xio/xerr.h>

int csi_reset (channel)
int channel;

DESCRIPTION

csi_reset terminates all pending commands for all processes on the Command Status Interface (CSI) port of the Image System Interface (ISI) board referenced by *channel*. A reset function code of 4 is placed on the DR11 function lines to reset the DR11 slave. *Channel* must have been opened with *csi_open*(3A).

csi_reset will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

csi_open(3A), csi_cmd(3A), csi_dstat(3A), csi_ustat(3A), csi_death(3A).
xcsi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

csi_status - read the CSI port DR11 status lines

SYNOPSIS

#include <sys/xio/xerr.h>

int csi_status (channel, stat)
int channel, *stat;

DESCRIPTION

csi_status returns the value of the Command Status Interface (CSI) port DR11 status lines which are directly connected to application specific external hardware through the Image System Interface (ISI) board referenced by *channel*. *Channel* must have been opened with *csi_open*(3A).

csi_status will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[ISI_REDUNDANT_REQ]	The specified *channel* is currently open by this process.
[ISI_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[ISI_CHANNEL_NOT_FOUND]	The specified *channel* is not present in the system.

SEE ALSO

csi_open(3A).
xcsi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

csi_ucan - cancel unsolicited status requests on a CSI port

SYNOPSIS

#include <sys/xio/xerr.h>

int csi_ucan (channel)
int channel;

DESCRIPTION

csi_ucan terminates all pending unsolicited status requests associated with the calling process on the Command Status Interface (CSI) port of the Image System Interface (ISI) board referenced by *channel*. *Channel* must have been opened with *csi_open*(3A).

csi_ucan will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

csi_open(3A), csi_ustat(3A).
xcsi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

csi_ustat, csi_ustat_nw - receive unsolicited status from a CSI port

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int csi_ustat (channel, sbuf, scnt, timeout, xfcnt)
int channel, scnt, timeout
char *sbuf;
int *xfcnt;

int csi_ustat_nw (channel, sbuf, scnt, timeout, xiosb, efn)
int channel, scnt, timeout, efn;
char *sbuf;
struct xiosb *xiosb;
```

DESCRIPTION

csi_ustat provides a mechanism to establish a receive buffer to catch unsolicited status over the Command Status Interface (CSI) port on the Image System Interface (ISI) board referenced by *channel*. *Channel* must have been opened with *csi_open*(3A).

In the event that an unsolicited or unclaimed status packet is received over the CSI port, the packet is received to the first unsolicited status buffer in the queue and the request associated with that buffer completes.

Only one process may have unsolicited status requests outstanding at a time. An appropriate status is returned if another process is gathering unsolicited status for the *channel*.

Sbuf is a pointer to the unsolicited status buffer and must begin on a longword boundary. *Scnt* is the size (in bytes) of *sbuf*. This value must be greater than or equal to 20 which is the packet header size.

Timeout is the time limit in 1/60 second intervals to receive an unsolicited status packet. Upon expiration, the request will be aborted and an appropriate status is returned. A value of 0 disables the timeout function.

Upon completion of the synchronous request, the integer pointed to by *xfcnt* is updated with the number of bytes received in the status packet.

csi_ustat_nw is the asynchronous version of *csi_ustat*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes received in the status buffer.

csi_ustat and *csi_ustat_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.

[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[ISI_CHANNEL_BUSY]	Another process is already collecting unsolicited status packets on this channel.
[BAD_DATA_BUFFER_ADDRESS]	*Sbuf* is not long-word aligned or points to a nonwritable memory space.
[BAD_DATA_BUFFER_SIZE]	The *scnt* is smaller than the header size of 20 bytes.
[PAGE_LOCK_FAILED]	Not enough physical memory for this request is available at this time.
[ISI_CANCELED]	The request was canceled with either *csi_ucan*(3A) or *csi_close*(3A).
[ISI_PARITY_ERROR]	A parity error occurred during the transfer.
[ISI_CYCLE_ERROR]	A hardware handshake error occurred during the transfer.
[ISI_TIMEOUT]	The *timeout* expired before the status packet was received.
[ISI_PROTOCOL_ERROR]	A firmware handshake error occurred during the transfer. This probably means that *scnt* was smaller than the actual size of the status packet.
[ISI_HARDWARE_CHECK]	A fatal status code was asserted on the DR11 status lines during the transfer.

SEE ALSO

intro(3A), csi_open(3A), csi_close(3A), csi_ucan(3A).
xcsi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

fg_alloc - allocate a frame grabber

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xfg.h>

int fg_alloc (fgno, info, excl)
int fgno, excl;
struct fg_info *info;
```

DESCRIPTION

fg_alloc allocates a frame grabber for an application to use. If *excl* is nonzero, the calling process has exclusive access to the frame grabber until it deallocates the frame grabber or the process exits. If *excl* is 0, another process may allocate the same frame grabber.

Fgno specifies the location of the frame grabber on the Shared Resource (SR) Bus. The frame grabber in the lowest SR Bus slot will be addressed by 0. This number is used with all subsequent calls.

Info points to an *fg_info* structure as defined in the header file **<sys/xio/xfg.h>**:

```
struct fg_info {
     int   i_slot;          /* frame grabber hardware slot number */
     long  *i_fb0_addr;     /* frame buffer 0 address */
     long  *i_fb1_addr;     /* frame buffer 1 address */
}
```

Info will be filled in upon return by *fg_alloc*. *I_fb0_addr* and *i_fb1_addr* point to the virtual addresses of frame buffers 0 and 1 (respectively) of the frame grabber. Each buffer consists of 256K longwords organized as an array of 512 x 512, 32-bit pixels. The format for each 32-bit pixel is as follows:

Byte	Description
0	red color component
1	green color component
2	blue color component
3	not used

fg_alloc will fail if one of the following are true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FG_NOT_PRESENT]	A frame grabber is not available for allocation.
[BAD_DATA_BUFFER_ADDRESS]	*Info* or *fgno* points to a nonwritable memory address.

[FG_BUSY]	Another user allocated the frame grabber for exclusive use. Or, this call requested exclusive use and another process allocated the frame grabber for nonexclusive use.
[FG_RESOURCE_ERROR]	Not enough virtual memory is available to map the frame buffer into the user's memory space.

SEE ALSO

fg_dealloc(3A).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

WARNINGS

Cooperation among processes must occur when sharing a frame grabber.

NAME

fg_blank - blank the output signal of the frame grabber

SYNOPSIS

#include <sys/xio/xerr.h>

int fg_blank (fgno, mode)
int fgno, mode;

DESCRIPTION

fg_blank forces the output video to be blanked when *mode* is nonzero. If *mode* is zero, the output video is sourced from one of the frame buffers.

Fgno specifies the location of the frame grabber on the Shared Resource (SR) Bus. The frame grabber in the lowest SR Bus slot will be addressed by 0.

fg_blank will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FG_NOT_OWNER]	The specified frame grabber was not allocated by the calling process.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fg_dealloc - deallocate a frame grabber

SYNOPSIS

#include <sys/xio/xerr.h>

int fg_dealloc (fgno)
int fgno;

DESCRIPTION

fg_dealloc deallocates a frame grabber. Any further reference to the frame grabber is disallowed. In addition, the frame buffer memory is unmapped from the calling process.

Fgno specifies the frame grabber location on the Shared Resource (SR) Bus. The frame grabber in the lowest SR Bus slot will be addressed by 0.

fg_dealloc will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FG_NOT_OWNER]	The specified frame grabber was not allocated by the calling process.

SEE ALSO

fg_alloc(3A).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fg_fbmode - set the mode of a frame buffer

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xfg.h>

int fg_fbmode (fgno, fbno, mode)
int fgno, fbno, mode;
```

DESCRIPTION

fg_fbmode sets the mode of a frame buffer to FG_DISPLAY, FG_GRAB, or FG_SNAP. If *fbno* is zero, the mode is set for frame buffer 0. Otherwise, the mode is set for frame buffer 1.

Fgno specifies the location of the frame grabber on the Shared Resource (SR) Bus. The frame grabber in the lowest SR Bus slot will be addressed by 0.

Mode FG_DISPLAY forces the specified frame buffer to continuously provide the output video in the selected format. If both frame buffers are in FG_DISPLAY_MODE, the frame buffer specified by *fg_priority*(3A) is displayed.

Mode FG_GRAB forces the digitized video signal to be continuously stored in the specified frame buffer.

Mode FG_SNAP will force the specified frame buffer into FG_GRAB mode for one frame of video; it will then change the mode for that frame buffer to FG_DISPLAY.

fg_fbmode will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FG_NOT_OWNER]	The specified frame grabber was not allocated by the calling process.
[FG_INVALID_MODE]	*Mode* was not specified as FG_DISPLAY, FG_GRAB, or FG_SNAP.

SEE ALSO

fg_priority(3A), fg_fbstat(3A).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fg_fbstat - get the mode of a frame buffer

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xfg.h>

int fg_fbstat (fgno, fbno, mode)
int fgno, fbno;
int *mode;
```

DESCRIPTION

fg_fbstat returns the mode of a frame buffer. The mode is returned in the location referenced by *mode*. *Mode* will be either FG_DISPLAY, FG_GRAB, or FG_SNAP. *fg_fbstat* determines when a frame was grabbed by polling until the mode FG_DISPLAY is returned.

Fgno specifies the frame grabber location on the Shared Resource (SR) Bus. The frame grabber in the lowest SR Bus slot will be addressed by 0.

fg_fbstat will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FG_NOT_OWNER]	The specified frame grabber was not allocated by the calling process.

SEE ALSO

fg_fbmode(3A).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fg_lut_in, fg_lut_out - load the lookup tables of a frame grabber

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xfg.h>

int fg_lut_in (fgno, table, color)
int fgno, color;
char table[];

int fg_lut_out (fgno, table, color, mode)
int fgno, color, mode;
char table[];
```

DESCRIPTION

fg_lut_in loads one or all of the three input lookup tables that translate 8-bit data between the analog-to-digital converters and the frame buffers. *Table* is a pointer to 256 bytes of data to be loaded into the input lookup tables. The data is loaded into the red, green, or blue lookup table by assigning *color* to FG_RED, FG_GREEN, or FG_BLUE. If *color* is assigned FG_ALL, the same data is loaded into all three lookup tables.

fg_lut_out works similarly to *fg_lut_in*, controlling the output lookup tables that translate the data between the frame buffers and the digital-to-analog converters. The output lookup tables differ from the input lookup tables in that four extra bits of information go into each output lookup table. The following figure shows the generated address for the output lookup tables:

11	10	9	8	7	...	0
Y ADDRESS (LSB 2 BITS)		X ADDRESS (LSB 2 BITS)		RASTER DATA		

The extra bits represent the least significant two bits of the X and Y positions for the pixel location being sent through the lookup tables. These extra four address lines are useful when the lookup tables loaded with a dither mapping are accessed. The total usable entries in the lookup table in this mode are 4K. *Mode* indicates if *table* contains 4K or 256 bytes. If *mode* is nonzero, the full 4K is written to the output lookup tables. If *mode* is zero, *table* is assumed to be 256 bytes and is replicated in the output lookup table so that the four position bits are treated as "don't cares."

Fgno specifies the frame grabber location on the Shared Resource (SR) Bus. The frame grabber in the lowest SR Bus slot will be addressed by 0.

fg_lut_in and *fg_lut_out* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.

[FG_NOT_OWNER]	The specified frame grabber was not allocated by the calling process.
[FG_INVALID_LUT]	*Color* is not a valid lookup table specifier.
[BAD_DATA_BUFFER_ADDRESS]	*Table* points to an invalid memory address.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

WARNINGS

The output lookup table may be partially loaded if an error occurs when the user's table is read.

NAME

fg_priority - determine frame buffer output priority of the frame grabber

SYNOPSIS

#include <sys/xio/xerr.h>

int fg_priority (fgno, fbno)
int fgno, fbno;

DESCRIPTION

fg_priority determines the source of video output when both frame buffers are in FG_DISPLAY mode. If *fbno* is zero, frame buffer 0 sources the video output. Otherwise, frame buffer 1 sources the video output.

Fgno specifies the frame grabber location on the Shared Resource (SR) Bus. The frame grabber in the lowest SR Bus slot will be addressed by 0.

fg_priority will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FG_NOT_OWNER]	The specified frame grabber was not allocated by the calling process.

SEE ALSO

fg_fbmode(3A).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fg_reset - force the frame grabber to a known state

SYNOPSIS

```
#include <sys/xio/xerr.h>

int fg_reset (fgno)
int fgno;
```

DESCRIPTION

fg_reset forces the frame grabber to the following state:

Video output is blanked.

Frame buffer 0 has display priority over frame buffer 1.

Both frame buffers are set to FG_DISPLAY mode.

Input and Output video is National Television Systems Committee (NTSC).

Input and Output Lookup Tables are loaded with a transparent (1:1) pattern.

Fgno specifies the frame grabber location on the Shared Resource (SR) Bus. The frame grabber in the lowest SR Bus slot will be addressed by 0.

fg_reset will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FG_NOT_OWNER]	The specified frame grabber was not allocated by the calling process.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fg_size - determine the frame grabber window size

SYNOPSIS

```
#include <sys/xio/xerr.h>

int fg_size (fgno, x, y)
int fgno;
int *x, *y;
```

DESCRIPTION

fg_size returns the window size needed to completely display the video image. Currently, two sizes are possible: 482 rows by 512 columns and 512 rows by 512 columns.

Fgno specifies the frame grabber location on the Shared Resource (SR) Bus. The frame grabber in the lowest SR Bus slot will be addressed by 0.

The integer pointed to by *x* will be set to the number of columns, and the integer pointed to by *y* will be set to the number of rows.

fg_size will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FG_NOT_OWNER]	The specified frame grabber was not allocated by the calling process.

SEE ALSO

fg_viw_start(3A).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fg_video_in, fg_video_out - select the video signal types for I/O

SYNOPSIS

```
#include <sys/xio/xerr.h>

int fg_video_in (fgno, mode)
int fgno, mode;

int fg_video_out (fgno, mode)
int fgno, mode;
```

DESCRIPTION

fg_video_in selects the video signal type for the frame grabber to digitize. If *mode* is zero, the Red Green Blue (RGB) video input is digitized. If *mode* is nonzero, the National Television Systems Committee (NTSC) video input is digitized.

fg_video_out selects the video signal type generated by the frame grabber. If *mode* is zero, an RGB video signal is generated. If *mode* is nonzero, an NTSC video signal is generated.

Fgno specifies the frame grabber location on the Shared Resource (SR) Bus. The frame grabber in the lowest SR Bus slot will be addressed by 0.

fg_video_in and *fg_video_out* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FG_NOT_OWNER]	The specified frame grabber was not allocated by the calling process.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fg_viw_start, fg_viw_stop - start and stop video in a window

SYNOPSIS

#include <sys/xio/xerr.h>

int fg_viw_start (fgno, wno, x, y, mode)
int fgno, wno, x, y, mode;

int fg_viw_stop (fgno)
int fgno;

DESCRIPTION

fg_viw_start forces the frame grabber into a double-buffered sequence of storing frames coordinated with the Integrated Frame Buffer (IFB) graphics processor displaying the stored frames. *Wno* is the window in which the captured video will be displayed. The upper-left corner of the video image will be displayed at the window-relative offset (*x*,*y*). If *mode* is zero, the graphics processor will display 8 raster planes. If *mode* is nonzero, 9 planes will be displayed. Video will continue to be displayed until stopped by *fg_viw_stop*.

Fgno specifies the frame grabber location on the Shared Resource (SR) Bus. The frame grabber which resides in the lowest SR Bus slot will be addressed by 0.

fg_viw_start and *fg_viw_stop* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FG_NOT_OWNER]	The specified frame grabber was not allocated by the calling process.
[FG_INVALID_WINDOW]	*Wno* references a nonexistent window.
[FG_BUSY]	A frame grabber is already running in this mode.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

WARNINGS

No other frame grabber commands should be called between *fg_viw_start* and *fg_viw_stop*.

NAME

fpe_cancel_dma - cancel write request to an FPE coprocessor

SYNOPSIS

```
#include <sys/xio/xerr.h>

int fpe_cancel_dma (fpeno)
int fpeno;
```

DESCRIPTION

fpe_cancel_dma stops the active transfer to a Floating-Point Engine (FPE), and aborts transfers waiting to be started. *Fpeno* is the number of the FPE obtained by *fpe_coproc_alloc*(3A).

fpe_cancel_dma will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FPE_NOT_PRESENT]	*Fpeno* specifies an FPE that is not in the system.
[FPE_NOT_OWNER]	*Fpeno* specifies an FPE that is not currently allocated by the process.

SEE ALSO

fpe_coproc_alloc(3A), fpe_write_dma(3A).
xfpe(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fpe_coproc_alloc - allocate an FPE coprocessor

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xfpe.h>

int fpe_coproc_alloc (fpeno, info)
int *fpeno;
struct fpe_info *info;
```

DESCRIPTION

fpe_coproc_alloc allocates a Floating-Point Engine (FPE) for application-specific numeric operations. A process which successfully allocates an FPE has sole control until it deallocates the FPE, execs, or exits.

Fpeno points to a location that is updated with the number of the allocated FPE. A process uses this number with subsequent FPE functions to access the allocated FPE.

Info points to an *fpe_info* structure filled in by *fpe_coproc_alloc*. As defined in the header file **<sys/xio/xfpe.h>**, the structure has the following members:

```
int    i_slot;        /* FPE hardware slot number */
int    *i_fifo;       /* FPE fifo register pointer */
int    *i_status;     /* FPE status register pointer */
```

The access mode for the *i_fifo* pointer is read/write, and the access mode for the *i_status* pointer is read-only.

fpe_coproc_alloc will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FPE_NOT_PRESENT]	A floating-point processor is not available for allocation.
[BAD_DATA_BUFFER_ADDRESS]	*Info* points to a nonwritable memory address.

SEE ALSO

fpe_coproc_dealloc(3A).
xfpe(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fpe_coproc_dealloc - deallocate an FPE coprocessor

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xfpe.h>

int fpe_coproc_dealloc (fpeno)
int fpeno;
```

DESCRIPTION

fpe_coproc_dealloc relinquishes control of a Floating-Point Engine (FPE) coprocessor which was previously allocated with *fpe_coproc_alloc*(3A). *Fpeno* is the number of the FPE to be deallocated.

fpe_coproc_dealloc will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FPE_NOT_PRESENT]	*Fpeno* specifies an invalid FPE.
[FPE_NOT_OWNER]	A process attempts to deallocate an FPE it did not previously allocate.

SEE ALSO

fpe_coproc_alloc(3A).
xfpe(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fpe_did_load - load an FPE coprocessor image

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xfpe.h>

int fpe_did_load (fpeno, file, did)
int fpeno, *did;
char *file;
```

DESCRIPTION

fpe_did_load loads an application-specific microcode image into a Floating-Point Engine (FPE) coprocessor. Obtained with *fpe_coproc_alloc*(3A), *fpeno* is the number of the FPE to be loaded.

File points to the path name of the file containing the FPE microcode image to be loaded. Before loading the microcode image into the FPE, *fpe_did_load* checks the process's read access permission for the file and ensures that the file is not currently opened for write operations. Then, *fpe_did_load* marks the file as being opened for execution; this ensures that a process cannot write to the file while it is being loaded. The driver clears the execute status when the microcode image is unloaded.

The driver supports a maximum of NDID—1 loaded microcode images.

Upon successful completion, *fpe_did_load* updates the *did* pointer location with an identification number the driver associates with the loaded microcode image. *fpe_did_unload*(3A) uses this number to unload the proper microcode image.

fpe_did_load will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FPE_NOT_PRESENT]	*Fpeno* specifies an FPE that is not in the system.
[FPE_NOT_OWNER]	*Fpeno* specifies an FPE that is not currently allocated by the process.
[FPE_DID_FILE_NOT_FOUND]	*File* points to an invalid file name.
[FPE_DID_NO_ACCESS_PRIV]	*File* specifies a file that lacks read permission for the process.
[FPE_DID_FILE_BUSY]	*File* specifies a file that is currently opened for writing.
[FPE_DID_DEVICE_FULL]	The process attempts to exceed the maximum number of loaded microcode images supported.

[FPE_DID_BAD_FILE]	A failure occurs while reading the microcode file, or the file's header is invalid.
[FPE_DID_NOSPACE]	The microcode image is larger than the available image memory in the FPE.
[FPE_DID_INIT_FAILURE]	The FPE fails to initialize properly after loading the microcode image.

SEE ALSO

fpe_coproc_alloc(3A), fpe_did_unload(3A).
xfpe(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fpe_did_unload - unload an FPE coprocessor image

SYNOPSIS

#include <sys/xio/xerr.h>

int fpe_did_unload (fpeno, did)
int fpeno, did;

DESCRIPTION

fpe_did_unload unloads a Floating-Point Engine (FPE) microcode image which was loaded using *fpe_did_load*(3A). *Fpeno*, returned by *fpe_coproc_alloc*(3A), is the number of the FPE to be unloaded by the driver.

Did is the identification number of the microcode image to be unloaded obtained by *fpe_did_load*(3A).

fpe_did_unload will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[FPE_NOT_PRESENT]	*Fpeno* specifies an invalid FPE.
FPE_NOT_OWNER	*Fpeno* specifies an FPE that is not currently allocated by the process.
[FPE_DID_OUT_OF_RANGE]	*Did* specifies a microcode image greater than the maximum number supported by the driver.
[FPE_DID_NONEXISTENT]	*Did* specifies a microcode image that is not currently loaded in the FPE.

SEE ALSO

fpe_coproc_alloc(3A), fpe_did_load(3A).
xfpe(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

fpe_write_dma, fpe_write_dma_nw - write data to an FPE coprocessor

SYNOPSIS

```
#include <sys/types.h>
#include <sys/immu.h>
#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>
#include <sys/xio/xfpe.h>

int fpe_write_dma (fpeno, dbuf, dcnt, timeout, xfcnt)
int fpeno, dcnt, timeout;
char *dbuf;
int *xfcnt;

int fpe_write_dma_nw (fpeno, dbuf, dcnt, timeout, xiosb, efn)
int fpeno, dcnt, timeout, efn;
char *dbuf;
struct xiosb *xiosb;
```

DESCRIPTION

fpe_write_dma transfers data to a Floating-Point Engine (FPE) coprocessor. *Fpeno*, obtained by *fpe_coproc_alloc*(3A), is the number of the FPE which receives the data.

Dbuf points to the buffer containing the data to be written to the FPE. *Dcnt* is the size (in bytes) of *dbuf*. The buffer must be long-word aligned and have a size that is a multiple of 4 bytes.

Timeout is the number of 1/60 second intervals *fpe_write_dma* waits before halting an active transfer. A value of 0 disables the timeout feature.

Upon completion of the synchronous request, the integer pointed to by *xfcnt* is updated with the number of bytes successfully written to the FPE.

fpe_write_dma_nw is the asynchronous version of *fpe_write_dma*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes successfully written to the FPE.

fpe_write_dma and *fpe_write_dma_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[FPE_NOT_PRESENT]	*Fpeno* specifies an FPE that is not in the system.
[FPE_NOT_OWNER]	*Fpeno* specifies an FPE that is not currently allocated by the process.

[BAD_DATA_BUFFER_ADDRESS]	*Dbuf* points to buffer that is not long-word aligned or is to an invalid memory space.
[BAD_DATA_BUFFER_SIZE]	*Dcnt* is not a multiple of 4 bytes.
[PAGE_LOCK_FAILED]	The system is unable to lock down all the pages needed to satisfy the request.
[FPE_DMA_CANCELED]	*fpe_dma_cancel*(3A) canceled the request.
[FPE_DMA_TIMEOUT]	A timeout occurred before the request completed.

SEE ALSO

intro(3A), fpe_coproc_alloc(3A), fpe_cancel_dma(3A).
xfpe(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

gpib_cancel - cancel all outstanding requests on a GPIB channel

SYNOPSIS

#include <sys/xio/xerr.h>

int gpib_cancel (channel)
int channel;

DESCRIPTION

gpib_cancel stops any active operation on the specified General Purpose Interface Bus (GPIB) *channel*. In addition, all outstanding control, transfer, and interrupt requests are canceled.

gpib_cancel will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.

SEE ALSO

gpib_open(3A), gpib_read(3A), gpib_write(3A), gpib_service(3A), gpib_cmd(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_clear - clear a GPIB channel or device

SYNOPSIS

```
#include <sys/xio/xerr.h>

int gpib_clear (channel, ldev, lcnt)
int channel, lcnt;
char *ldev;
```

DESCRIPTION

gpib_clear allows a user to clear the entire General Purpose Interface Bus (GPIB) *channel* or selected devices on *channel.* An entire channel is cleared by setting *lcnt* to 0. If specific devices are to be cleared, *ldev* points to an array GPIB primary addresses for the device and *lcnt* equals the number of addresses in the array.

To clear *channel,* the driver puts the system controller in an active state and issues the GPIB message Device Clear (DCL).

To clear specific devices, the driver puts the system controller in an active state and sends the following GPIB messages:

UNL	Unlisten
LAG	Listen Address Group
SDC	Selected Device Clear

gpib_clear will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.
[GPIB_DEVICE_INVALID]	One or more of the specified primary addresses are incorrect.
[BAD_DATA_BUFFER_ADDRESS]	The primary address pointer *ldev* is invalid.
[BAD_DATA_BUFFER_SIZE]	The device count *lcnt* is incorrect.

SEE ALSO

gpib_open(3A), gpib_reset(3A), gpib_cmd(3A), gpib_cancel(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_close - close a GPIB channel

SYNOPSIS

```
#include <sys/xio/xerr.h>

int gpib_close (channel)
int channel;
```

DESCRIPTION

gpib_close frees the General Purpose Interface Bus (GPIB) *channel* that was previously allocated with *gpib_open*(3A). The close cancels all outstanding control, transfer, and interrupt requests queued for the channel. Current transfers are also aborted.

gpib_close will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.

SEE ALSO

gpib_open(3A), gpib_cancel(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_cmd, gpib_cmd_nw - send commands to a GPIB channel

SYNOPSIS

```
#include <sys/types.h>
#include <sys/immu.h>
#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>
#include <sys/xio/xgpib.h>

int gpib_cmd (channel, cmd, ccnt, timeout)
int channel, ccnt, timeout;
char *cmd;

int gpib_cmd_nw (channel, cmd, ccnt, timeout, xiosb, efn)
int channel, ccnt, timeout, efn;
char *cmd;
struct xiosb *xiosb;
```

DESCRIPTION

gpib_cmd sends device-specific General Purpose Interface Bus (GPIB) command sequences directly to the GPIB *channel*. The controller is placed in the active state before sending the commands and remains in the active state upon completion.

Cmd points to an array of GPIB messages, while *ccnt* reflects the number of byte messages to be transferred. The user is responsible for ensuring the validity of GPIB messages, listener addresses, and talker addresses.

If an abort timeout is desired, *timeout* contains the number of 1/60 second intervals the driver waits before aborting the request. A value of 0 disables the timeout mechanism.

gpib_cmd_nw is the asynchronous version of *gpib_cmd*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transmitted.

gpib_cmd and *gpib_cmd_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.

[GPIB_CANCELED]	The request was canceled by *gpib_cancel*(3A).
[GPIB_TIMEOUT]	A timeout occurred before the request completed.
[BAD_DATA_BUFFER_ADDRESS]	*Cmd* points to an invalid memory address.
[BAD_DATA_BUFFER_SIZE]	The command count *ccnt* is incorrect.

SEE ALSO

intro(3A), gpib_read(3A), gpib_write(3A), gpib_cancel(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

gpib_local - return a GPIB device to local control

SYNOPSIS

```
#include <sys/xio/xerr.h>

int gpib_local (channel, ldev, lcnt)
int channel, lcnt;
char *ldev;
```

DESCRIPTION

gpib_local returns specific devices or all devices on the General Purpose Interface Bus (GPIB) *channel* to local control. *Ldev* points to an array of primary addresses corresponding to the GPIB devices that are to switch to local operation. *Lcnt* is the number of devices to be addressed. If *lcnt* is 0, all devices on the GPIB *channel* return to local control.

All devices connected to a GPIB channel are returned to local control by deasserting the GPIB Remote Enable (REN) signal.

gpib_local returns specific devices to local operation by putting the system controller in an active state and sending the following GPIB messages:

UNL	Unlisten
LAG	Listen Address Group
GTL	Go To Local

gpib_local will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.
[GPIB_DEVICE_INVALID]	One or more of the specified primary addresses are incorrect.
[BAD_DATA_BUFFER_ADDRESS]	*Ldev* points to an invalid memory address.
[BAD_DATA_BUFFER_SIZE]	The device count *lcnt* is incorrect.

SEE ALSO

gpib_open(3A), gpib_remote(3A), gpib_lockout(3A), gpib_cmd(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_lockout - issue a local lockout to a GPIB channel

SYNOPSIS

#include <sys/xio/xerr.h>

int gpib_lockout (channel)
int channel;

DESCRIPTION

gpib_lockout puts General Purpose Interface Bus (GPIB) devices in a local lockout mode, thereby disabling local control. *Channel* is the GPIB target for a local lockout operation.

The system controller is put in an active state before a local lockout operation is performed. The global Local Lock Out (LLO) message is issued on the GPIB *channel.*

If the GPIB Remote Enable (REN) condition is not active (the GPIB channel is not in a remote state), a device's local lockout is not in effect. However, once the REN condition is active, a device that was previously sent a LLO message proceeds to the lockout state when addressed.

gpib_lock will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.

SEE ALSO

gpib_open(3A), gpib_remote(3A), gpib_local(3A), gpib_cmd(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_open - open a GPIB channel

SYNOPSIS

```
#include <sys/xio/xerr.h>

int gpib_open (channel)
int channel;
```

DESCRIPTION

gpib_open allocates the General Purpose Interface Bus (GPIB) *channel* for GPIB operations. A process which successfully opens a channel receives sole control until it closes the channel, exits, or execs.

gpib_open will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_IN_USE]	*Channel* is currently open by another process.
[GPIB_NOT_FOUND]	GPIB *channel* is not in the system.

SEE ALSO

gpib_close(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_ppconf - configure the parallel poll response of a GPIB device

SYNOPSIS

#include <sys/xio/xerr.h>

int gpib_ppconf (channel, ldev, mask)
int channel;
char *ldev, mask;

DESCRIPTION

gpib_ppconf configures a device connected to the General Purpose Interface Bus (GPIB) *channel* with a specific parallel poll response. *Ldev* points to the primary address of the device, while *mask* contains the response. The format of the response mask, as defined in the IEEE 488 standard, is given below.

Bit Field	Description
0-2	binary coded GPIB data line on which the device is to respond
3	logic level of the response

gpib_ppconf configures a GPIB device's parallel poll response by putting the system controller in an active state and issuing the following sequence of messages.

UNL	Unlisten
LAG	Listen Address Group
PPC	Parallel Poll Configure
PPE	Parallel Poll Enable
UNL	Unlisten

gpib_ppconf will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.
[GPIB_DEVICE_INVALID]	The primary address of the specified device is incorrect.
[BAD_DATA_BUFFER_ADDRESS]	The primary address pointer *ldev* points to an invalid memory address.

SEE ALSO

gpib_open(3A), gpib_ppreq(3A), gpib_ppuconf(3A), gpib_cmd(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_ppreq - perform a parallel poll of a GPIB channel

SYNOPSIS

#include <sys/xio/xerr.h>

int gpib_ppreq (channel, timeout, poll)
int channel, timeout;
char *poll;

DESCRIPTION

gpib_ppreq conducts parallel polls of the General Purpose Interface Bus (GPIB) *channel*. The address pointed to by *poll* contains the parallel poll response upon completion.

If an abort timeout is desired, *timeout* contains the number of 1/60 second intervals that the driver waits before aborting the request. A value of 0 disables the timeout mechanism.

gpib_ppreq will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.
[GPIB_TIMEOUT]	The current request aborted before the parallel poll response was received from the GPIB *channel*.
[BAD_DATA_BUFFER_ADDRESS]	*Poll* points to an invalid memory address.
[GPIB_HARDWARE_CHECK]	A hardware error was detected during the parallel poll of the specified GPIB *channel*. For example, none of the devices had been previously configured for parallel polling.

SEE ALSO

gpib_open(3A), gpib_ppconf(3A), gpib_ppuconf(3A), gpib_service(3A), gpib_spreq(3A), gpib_cmd(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_ppuconf - unconfigure a GPIB device's parallel poll response

SYNOPSIS

```
#include <sys/xio/xerr.h>

int gpib_ppuconf (channel, ldev, lcnt)
int channel, lcnt;
char *ldev;
```

DESCRIPTION

gpib_ppuconf disables a General Purpose Interfaces Bus (GPIB) device's ability to respond to a parallel poll request on *channel*. *Ldev* points to an array of primary addresses of the GPIB devices to be unconfigured. *Lcnt* is the number of devices. If all devices on a GPIB *channel* are to be unconfigured, *lcnt* is 0.

After putting the system controller in its active state, *gpib_ppuconf* unconfigures specific GPIB devices with the following messages.

UNL	Unlisten
LAG	Listen Address Group
PPC	Parallel Poll Configure
PPD	Parallel Poll Disable
UNL	Unlisten

If a global Parallel Poll Unconfigure (PPU) is indicated, the system controller is switched to an active state, and the PPU message is issued on the GPIB *channel*.

gpib_ppuconf will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.
[GPIB_DEVICE_INVALID]	One or more of the specified primary addresses are incorrect.
[BAD_DATA_BUFFER_ADDRESS]	*Ldev* points to an invalid memory address.
[BAD_DATA_BUFFER_SIZE]	The number of devices specified with *lcnt* is incorrect.

SEE ALSO

gpib_open(3A), gpib_ppconf(3A), gpib_ppreq(3A), gpib_cmd(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_read, gpib_read_nw - read data from a GPIB device

SYNOPSIS

```
#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>

int gpib_read (channel, tdev, tcnt, dbuf, dcnt, timeout, eoi, xfcnt)
int channel, tcnt, dcnt, timeout;
char *tdev, *dbuf, *eoi;
int *xfcnt;

int gpib_read_nw (channel, tdev, tcnt, dbuf, dcnt, timeout, eoi,
                  xiosb, efn)
int channel, tcnt, dcnt, timeout, efn;
char *tdev, *dbuf, *eoi;
struct xiosb *xiosb;
```

DESCRIPTION

gpib_read allows data to be received from a General Purpose Interface Bus (GPIB) device on *channel*. The primary address from which to read is pointed to by *tdev*.

Dbuf is a pointer to the buffer to receive the data. The size of the buffer is specified (in bytes) by *dcnt*.

Setting *tcnt* to 1 will initialize the specified GPIB device as a talker before the data transfer begins. Setting *tcnt* to 0 indicates the device has already been addressed to talk by a previous *gpib_read*, and no device initialization will occur.

If an abort timeout is desired, *timeout* contains the number of 1/60 second intervals the driver waits before aborting the request. A value of 0 disables the timeout mechanism.

The address pointed to by *eoi* is updated with a nonzero value if the GPIB End Or Identify (EOI) condition was active upon completion. Conversely, a value of 0 indicates a deasserted EOI signal upon reception of the last data byte. If the driver detects an EOI condition before *dcnt* bytes are transferred to the buffer, the *eoi* status location is nonzero, and a successful completion status is returned.

Upon completion of the synchronous request, the integer pointed to by *xfcnt* contains the number of bytes transferred to *dbuf*.

gpib_read_nw is the asynchronous version of *gpib_read*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred to *dbuf*.

If a primary address is specified, the controller is put in an active state, and the following GPIB messages are sent over the specified channel.

UNL	Unlisten
UNT	Untalk
MTA	My Talk Address
LAG	Listen Group Address

The system controller is forced to the standby state, and data is read from the channel until *dcnt* bytes are received or the EOI condition is detected. If a primary address is not specified (*tcnt* is 0), the system controller is put in a standby state, and the read proceeds as described.

gpib_read and *gpib_read_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.
[GPIB_CANCELED]	The current request was canceled by *gpib_cancel*(3A).
[GPIB_DEVICE_INVALID]	The specified device's primary address was not correct.
[GPIB_TIMEOUT]	A timeout occurred before the request completed.
[GPIB_HARDWARE_CHECK]	A hardware error was detected during the request.
[BAD_DATA_BUFFER_ADDRESS]	Either *tdev* points to an invalid memory address, or *eoi*, *xfcnt*, or *dbuf* points to a nonwritable memory space.
[BAD_DATA_BUFFER_SIZE]	The data transfer size *dcnt* is too large.

SEE ALSO

intro(3A), gpib_open(3A), gpib_cmd(3A), gpib_cancel(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

gpib_remote - put a GPIB channel in a remote state

SYNOPSIS

#include <sys/xio/xerr.h>

int gpib_remote (channel)
int channel;

DESCRIPTION

gpib_remote activates the General Purpose Interface Bus (GPIB) Remote Enable (REN) signal on GPIB *channel*. This forces any GPIB device addressed to a remote state.

The system controller asserts the GPIB REN line to initiate the *gpib_remote* function. The system controller's state is not changed.

gpib_remote will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.

SEE ALSO

gpib_open(3A), gpib_lockout(3A), gpib_local(3A).
gpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_reset - conduct an IFC operation on a GPIB channel

SYNOPSIS

#include <sys/xio/xerr.h>

int gpib_reset (channel)
int channel;

DESCRIPTION

gpib_reset activates a GPIB Interface Clear (IFC) condition on *channel.* As defined in the IEEE 488 standard, the GPIB IFC signal is guaranteed to be asserted for a minimum of 100 microseconds. When this signal is cleared, the controller is put in a standby state.

gpib_reset will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.

SEE ALSO

gpib_open(3A), gpib_clear(3A), gpib_cancel(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_service, gpib_service_nw - request notification for a GPIB SRQ condition

SYNOPSIS

#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>

int gpib_service (channel)
int channel;

int gpib_service_nw (channel, xiosb, efn)
int channel, efn;
struct xiosb *xiosb;

DESCRIPTION

General Purpose Interface Bus (GPIB) devices generally request attention of the system controller through a Service Request (SRQ) interrupt. This condition can be detected on *channel* with *gpib_service.*

The *gpib_service* function returns only after the SRQ condition is detected. The initiating process waits indefinitely for an SRQ interrupt.

gpib_service_nw is the asynchronous version of *gpib_service*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure is not used.

gpib_service and *gpib_service_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.
[GPIB_CANCELED]	The currently queued request was canceled by *gpib_cancel*(3A).

SEE ALSO

intro(3A), gpib_open(3A), gpib_cancel(3A), gpib_ppreq(3A), gpib_spreq(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

gpib_spreq - conduct a serial poll of a GPIB device

SYNOPSIS

```
#include <sys/xio/xerr.h>

int gpib_spreq (channel, tdev, timeout, poll)
int channel, timeout;
char *tdev, *poll;
```

DESCRIPTION

The *gpib_spreq* function conducts a serial poll of a specific device on the General Purpose Interface Bus (GPIB) *channel*. *Tdev* points to the primary address of the device to be serially polled.

Upon completion, the status location pointed to by *poll* contains the serial poll response. If an error occurs during the request, the serial poll response is not valid.

Abort timeouts are enabled by setting *timeout* to the number of 1/60 second intervals the driver waits for the serial poll response. A value of 0 indicates a timeout is not desired.

To initiate the serial poll request, the system controller is put in an active state, and the following GPIB messages are sent on the GPIB *channel*:

UNL	Unlisten
UNT	Untalk
SPE	Serial Poll Enable
TAG	Talk Address Group

Following this message sequence, the controller enters a standby state and reads the serial poll response from the GPIB device. The controller is put in an active state, and the following GPIB messages are transmitted to ensure the device does not corrupt serial polls of other GPIB devices:

SPD	Serial Poll Disable
UNT	Untalk

gpib_spreq will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.
[GPIB_DEVICE_INVALID]	The specified primary address was incorrect.

[GPIB_TIMEOUT]	The current request aborted before the system controller received the GPIB serial poll response from the device.
[GPIB_HARDWARE_CHECK]	A hardware error occurred during the serial poll request. For example, the specified primary address may not correspond to any of the GPIB devices on *channel*.
[BAD_DATA_BUFFER_ADDRESS]	*Tdev* or *poll* points to an invalid memory address.

SEE ALSO

gpib_open(3A), gpib_service(3A), gpib_ppreq(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual*.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_trigger - trigger a GPIB device

SYNOPSIS

```
#include <sys/xio/xerr.h>

int gpib_trigger (channel, ldev, lcnt)
int channel, lcnt;
char *ldev;
```

DESCRIPTION

gpib_trigger performs a trigger operation on a General Purpose Interface Bus (GPIB) device. *Ldev* points to an array of GPIB primary addresses and *lcnt* specifies the number of devices to be triggered on the GPIB *channel*.

The system controller transmits the following messages to conduct a GPIB trigger:

UNL	Unlisten
LAG	Listen Address Group
GET	Group Execute Trigger

gpib_trigger will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.
[GPIB_DEVICE_INVALID]	The specified primary address is incorrect.
[BAD_DATA_BUFFER_ADDRESS]	*Ldev* points to an invalid memory address.
[BAD_DATA_BUFFER_SIZE]	The size of the primary address array specified by *lcnt* is too big.

SEE ALSO

gpib_open(3A), gpib_cmd(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

gpib_write, gpib_write_nw - write data to a GPIB device

SYNOPSIS

```
#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>

int gpib_write (channel, ldev, lcnt, dbuf, dcnt, timeout, eoi, xfcnt)
int channel, lcnt, dcnt, timeout, *xfcnt;
char *ldev, *dbuf, *eoi;

int gpib_write_nw (channel, ldev, lcnt, dbuf, dcnt, timeout, eoi,
                   xiosb, efn)
int channel, lcnt, dcnt, timeout, efn;
char *ldev, *dbuf, *eoi;
struct xiosb *xiosb;
```

DESCRIPTION

gpib_write writes data to the General Purpose Interface Bus (GPIB) device on *channel*. *Ldev* points to an array of primary addresses corresponding to the GPIB devices to receive data.

Dbuf is a pointer to the buffer to be transferred. The size of the buffer is specified (in bytes) by *dcnt*.

Lcnt reflects the number of devices to be initialized as listeners before the data transfer begins. A value of 0 indicates the GPIB devices have already been addressed to listen by a previous *gpib_write*. In this case, the write proceeds without device initialization.

If an abort timeout is desired, *timeout* contains the number of 1/60 second intervals the driver waits before aborting the request. A value of 0 disables the timeout mechanism.

If the value pointed to by *eoi* is nonzero, the GPIB End Or Identify (EOI) signal is asserted during the transfer of the last data byte in the buffer.

Upon completion of the synchronous request, the integer pointed to by *xfcnt* contains the number of bytes transferred from *dbuf*.

gpib_write_nw is the asynchronous version of *gpib_write*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred from *dbuf*.

If a primary address is specified, the controller is put in an active state, and the following GPIB messages are sent over the specified channel:

UNL	Unlisten
UNT	Untalk
TAG	Talk Address Group
MLA	My Listen Address

The system controller is forced to the standby state, and data is written to *channel* until *dcnt* bytes are transferred. If primary addresses are not specified, the system controller is simply put in a standby state, and the write proceeds as described.

gpib_write and *gpib_write_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[GPIB_OUT_OF_RANGE]	The specified *channel* is beyond the maximum allowed.
[GPIB_NOT_OPEN]	The specified *channel* is not open.
[GPIB_NOT_OWNER]	*Channel* is currently open by another process.
[GPIB_CANCELED]	The current request was canceled by *gpib_cancel*(3A).
[GPIB_DEVICE_INVALID]	A specified device's primary address was not correct.
[GPIB_TIMEOUT]	A timeout occurred before the request completed.
[GPIB_HARDWARE_CHECK]	A hardware error was detected during the request.
[BAD_DATA_BUFFER_ADDRESS]	Either *eoi*, *xfcnt*, *tdev*, or *dbuf* points to an invalid memory address.
[BAD_DATA_BUFFER_SIZE]	Either the data transfer size *dcnt* or the device count *lcnt* is too large.

SEE ALSO

intro(3A), gpib_open(3A), gpib_cmd(3A), gpib_cancel(3A).
xgpib(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

nlf_close - close an NLF channel

SYNOPSIS

```
#include <sys/xio/xerr.h>

int nlf_close (channel)
int channel;
```

DESCRIPTION

nlf_close unmaps any hardware registers associated with the specified Non-Linear Filter (NLF) *channel* and closes the *channel. Channel* must have been opened with *nlf_open*(3A).

nlf_close will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[NLF_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[NLF_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

nlf_open(3A).
xnlf(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

nlf_open - open an NLF channel

SYNOPSIS

#include <sys/xio/xerr.h>

int nlf_open (channel, base)
int channel, *base;

DESCRIPTION

nlf_open opens the specified Non-Linear Filter (NLF) *channel.* Only one process is allowed to open an NLF channel at a time. Each NLF board in the system is represented by a *channel* number so that *channel* 0 references the NLF board with the lowest Shared Resource (SR) Bus slot number. If the call is successful, *base* will contain the virtual base address of the specified NLF board. All filter parameters are available to the calling process through this mapping.

nlf_open will fail if one of the following is true:

[XIO_FAILURE]
The system does not contain the driver needed to support this request.

[NLF_REDUNDANT_REQ]	The specified *channel* is currently opened by this process.
[NLF_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[NLF_CHANNEL_NOT_FOUND]	The specified *channel* is not in the system.
[NLF_CHANNEL_BUSY]	The specified *channel* is currently opened by another process.

SEE ALSO

xnlf(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure is returned.

NAME

pdi_cancel - cancel outstanding asynchronous I/O on a PDI port

SYNOPSIS

```
#include <sys/xio/xerr.h>

int pdi_cancel (channel)
int channel;
```

DESCRIPTION

pdi_cancel terminates all requests pending on the Processed Data Interface (PDI) port of the Image System Interface (ISI) board referenced by *channel*. *Channel* must have been opened with *pdi_open*(3A).

pdi_cancel will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

pdi_open(3A), pdi_read(3A), pdi_write(3A), pdi_ifb(3A).
xpdi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

pdi_close - close a PDI port

SYNOPSIS

#include <sys/xio/xerr.h>

int pdi_close (channel)
int channel;

DESCRIPTION

pdi_close terminates all requests pending on the Processed Data Interface (PDI) port of the Image System Interface (ISI) board referenced by *channel* and closes the port. *Channel* must have been opened with *pdi_open*(3A).

pdi_close will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

pdi_open(3A).
xpdi(7A) in the *CLIX System Administrators Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

pdi_ifb, pdi_ifb_nw - move data from a PDI port to a window

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int pdi_ifb (channel, wno, xorg, yorg, xext, yext, timeout, xfcnt)
int channel, wno, xorg, yorg, xext, yext, timeout;
int *xfcnt;

int pdi_ifb_nw (channel, wno, xorg, yorg, xext, yext, timeout,
                xiosb, efn)
int channel, wno, xorg, yorg, xext, yext, timeout, efn;
struct xiosb *xiosb;
```

DESCRIPTION

pdi_ifb provides a mechanism for moving data directly from the Processed Data Interface (PDI) port of the Image System Interface (ISI) board referenced by *channel* to the specified window region on an Intergrated Frame Buffer (IFB) graphics board. *Channel* must have been opened with *pdi_open*(3A).

Data is transferred to the window specified by *wno*. *Xorg* and *yorg* indicate the window relative origin of the region to be filled. *Xext* and *yext* indicate the extents of the region to be filled. Both values must be a multiple of 32.

Timeout indicates the time limit in 1/60 second intervals for the data transfer. Any transfer that takes longer is aborted and an appropriate status is returned. A *timeout* value of 0 disables the timeout function.

Upon completion of the synchronous request, *xfcnt* indicates the number of bytes transferred.

pdi_ifb_nw is the asynchronous version of *pdi_ifb*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred.

pdi_ifb and *pdi_ifb_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[WINDOW_NONEXISTENT]	The specified window does not exist.
[ISI_INVALID_PARMS]	The origins and extents do not describe a valid nonzero region or the extents are not a multiple of 32.

[ISI_CANCELED]	The request was canceled with *pdi_cancel*(3A) or *pdi_close*(3A).
[ISI_PARITY_ERROR]	A parity error occurred on the transfer.
[ISI_CYCLE_ERROR]	A hardware handshake error occurred on the transfer.
[ISI_TIMEOUT]	The timeout occurred before the transfer completed.

SEE ALSO

intro(3A), pdi_open(3A), pdi_close(3A), pdi_setup(3A), pdi_cancel(3A). xpdi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

pdi_open - open a PDI port

SYNOPSIS

```
#include <sys/xio/xerr.h>

int pdi_open (channel)
int channel;
```

DESCRIPTION

pdi_open opens the Processed Data Interface (PDI) port on the Image System Interface (ISI) board referenced by *channel*. Only one process is allowed to open a PDI port at a time. Each ISI board in the system is represented by a *channel* number such that *channel* 0 references the ISI board with the lowest Shared Resource Bus slot number.

pdi_open will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[ISI_REDUNDANT_REQ]	The specified *channel* is currently open by this process.
[ISI_CHANNEL_BUSY]	The specified *channel* is currently open by another process.
[ISI_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[ISI_CHANNEL_NOT_FOUND]	The specified *channel* is not present in the system.

SEE ALSO

xpdi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

pdi_read, pdi_read_nw - read data from a PDI port into memory

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int pdi_read (channel, dbuf, dcnt, timeout, xfcnt)
int channel, dcnt, timeout;
char *dbuf;
int *xfcnt;

int pdi_read_nw (channel, dbuf, dcnt, timeout, xiosb, efn)
int channel, dcnt, timeout, efn;
char *dbuf;
struct xiosb *xiosb;
```

DESCRIPTION

pdi_read and *pdi_read_nw* provide a mechanism for reading directly from the Processed Data Interface (PDI) port on the Image System Interface (ISI) board referenced by *channel*. *Channel* must have been opened with *pdi_open*(3A).

Dbuf points to the data buffer. The buffer must begin on a long_word boundary. *Dcnt* contains the byte count to be transferred and must be a multiple of the *linewidth* specified in *pdi_setup*(3A).

Timeout indicates the time limit in 1/60 second intervals for the data transfer. Any transfer that takes longer is aborted and an appropriate status is returned. A *timeout* value of 0 disables the timeout function.

Upon completion of the synchronous request, *xfcnt* indicates the number of bytes transferred.

pdi_read_nw is the asynchronous version of *pdi_read*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred.

pdi_read and *pdi_read_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[BAD_DATA_BUFFER_ADDRESS]	The data buffer is not long-word aligned or points to a nonwritable memory address.

[BAD_DATA_BUFFER_COUNT]	The byte count is not a multiple of *linewidth* from *pdi_setup*(3A).
[PAGE_LOCK_FAILED]	Not enough physical memory for this request is available at this time.
[ISI_CANCELED]	The request was canceled with *pdi_cancel*(3A) or *pdi_close*(3A).
[ISI_PARITY_ERROR]	A parity error occurred on the transfer.
[ISI_CYCLE_ERROR]	A hardware handshake error occurred on the transfer.
[ISI_TIMEOUT]	The *timeout* expired before the transfer completed.

SEE ALSO

intro(3A), pdi_open(3A), pdi_close(3A), pdi_setup(3A), pdi_cancel(3A). xpdi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

sif_mem_pipe, sif_mem_pipe_nw - transfer data from memory to pipe

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int sif_mem_pipe (channel, dbuf, dcnt, timeout, xfcnt)
int channel, dcnt, timeout, *xfcnt;
char *dbuf;

int sif_mem_pipe_nw (channel, dbuf, dcnt, timeout, xiosb, efn)
int channel, dcnt, timeout, efn;
char *dbuf;
struct xiosb *xiosb;
```

DESCRIPTION

sif_mem_pipe and *sif_mem_pipe_nw* provide a mechanism for transferring data directly from virtual memory to the raster processing pipeline. The specified Scanner Interface (SIF) *channel* will move the data. *Channel* must have been opened with *sif_open*(3A).

Dbuf points to the buffer data is being transferred from. The buffer must begin on a long word boundary. *Dcnt* indicates the byte count to be transferred and must be a multiple of the *linewidth* specified in *sif_setup*(3A).

Timeout indicates the time limit in 1/60-second intervals for the data transfer. Any transfer that takes longer is aborted and an appropriate status is returned. A *timeout* value of zero disables the timeout function.

Upon completion of the synchronous request, *xfcnt* indicates the number of bytes transferred.

sif_mem_pipe_nw is the asynchronous version of *sif_mem_pipe*, providing the same capability without waiting the request to complete. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated when the request completes (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred.

sif_mem_pipe and *sif_mem_pipe_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[SIF_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[SIF_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

[BAD_DATA_BUFFER_ADDRESS]	The data buffer is either not long word aligned or points to an invalid memory space.
[BAD_DATA_BUFFER_COUNT]	The byte count is not a multiple of *linewidth* from *sif_setup*(3A).
[PAGE_LOCK_FAILED]	Not enough physical memory for this request is available at this time.
[SIF_CANCELED]	The request was canceled with *sif_cancel*(3A) or *sif_close*(3A).
[SIF_TIMEOUT]	The *timeout* expired before the transfer completed.

SEE ALSO

intro(3A), sif_open(3A), sif_close(3A), sif_setup(3A), sif_cancel(3A).
xsif(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the XIO system accepts the asynchronous request, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

pdi_setup - establish parameters for a PDI port

SYNOPSIS

#include <sys/xio/xerr.h>

int pdi_setup (channel, resolution, linewidth, write_valid)
int channel, resolution, linewidth, write_valid;

DESCRIPTION

pdi_setup establishes transfer parameters for the Processed Data Interface (PDI) port on the Image System Interface (ISI) board referenced by *channel*. *Channel* must have been opened with *pdi_open*(3A).

Resolution and *linewidth* are used together to determine what data is rejected by hardware on a *pdi_read*(3A). This should reduce memory and time requirements. The accepted data consists of the first *linewidth* bytes from each consecutive set of *resolution* bytes. All other data is rejected. If the concepts of *resolution* and *linewidth* do not apply, both should be set to 1. Possible values are 1, 32, 64, 128, 256, 512, and 1024. *Linewidth* must always be less than or equal to *resolution*. The default value for both parameters is 1.

The *write_valid* flag asserts a hardware signal to be interpreted by the external hardware.

pdi_setup will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[ISI_INVALID_PARMS]	The values for *resolution* and *linewidth* do not meet the stated requirements.

SEE ALSO

pdi_open(3A).
xpdi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

pdi_write, pdi_write_nw - write data from memory to a PDI port

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int pdi_write (channel, dbuf, dcnt, timeout, xfcnt)
int channel, dcnt, timeout;
char *dbuf;
int *xfcnt;

int pdi_write_nw (channel, dbuf, dcnt, timeout, xiosb, efn)
int channel, dcnt, timeout, efn;
char *dbuf;
struct xiosb *xiosb;
```

DESCRIPTION

pdi_write provides a mechanism for writing directly to the Processed Data Interface (PDI) port on the Image System Interface (ISI) board referenced by *channel*. *Channel* must have been opened with *pdi_open*(3A).

Dbuf points to the data buffer. The buffer must begin on a long-word boundary. *Dcnt* contains the number of bytes to be transferred.

Timeout indicates the time limit in 1/60 second intervals for the data transfer. Any transfer that takes longer is aborted and an appropriate status is returned. A *timeout* value of 0 disables the timeout function.

Upon completion of the synchronous request, *xfcnt* indicates the number of bytes transferred.

pdi_write_nw is the asynchronous version of *pdi_write*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred.

pdi_write and *pdi_write_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[ISI_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[BAD_DATA_BUFFER_ADDRESS]	The data buffer is not long-word aligned or points to an invalid memory space.
[PAGE_LOCK_FAILED]	Not enough physical memory for this request is available at this time.
[ISI_CANCELED]	The request was canceled with *pdi_cancel*(3A) or *pdi_close*(3A).

[ISI_PARITY_ERROR]	A parity error occurred on the transfer.
[ISI_CYCLE_ERROR]	A hardware handshake error occurred on the transfer.
[ISI_TIMEOUT]	The *timeout* expired before the transfer completed.

SEE ALSO

intro(3A), pdi_open(3A), pdi_close(3A), pdi_setup(3A), pdi_cancel(3A). xpdi(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to one of the above failure codes if unsuccessful.

NAME

plt_ctrl, plt_ctrl_nw - send a control word to the parallel port

SYNOPSIS

```
#include <sys/types.h>
#include <sys/immu.h>
#include <sys/pop.h>
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int plt_ctrl (interface, status, ctrl, timeout, pulse)
int interface, status, ctrl, timeout, pulse;

int plt_ctrl_nw (interface, status, ctrl, timeout, pulse, xiosb, efn)
int interface status, ctrl, timeout, pulse, efn;
struct xiosb *xiosb;
```

DESCRIPTION

plt_ctrl provides a mechanism to send a single word through the control register on the parallel port. *Interface* is one of the following values defined in **<sys/pop.h>**: CENTRONICS, VERSATEC, or INTERGRAPH_DIFF. These values define the signal mapping for the target device. The user provides the three-bit mask, *status*, that determines which incoming signals should be considered when testing for the clear-to-send condition. Bits that are set in the mask indicate that the corresponding signals received from the device should be asserted. The format of the status word is as follows:

2	1	0
READY H	NO ERROR H	ONLINE H

Ctrl is a 16-bit word that will be sent to the device (bits 0-7 are data; bits 8-15 are control). The control bits are as follows:

15	14	13	12	11	10	9	8
NOT USED	RESET H	RFFED H	REOTR H	RLTER H	CLEAR H	PICLK H	PRINT H

Pulse indicates whether bit 8 (PRINT H) is toggled as a control bit or interpreted as a data bit.

Timeout indicates the time limit in 1/60 second intervals to wait for the device to become ready before aborting the request. This value is limited to 32767.

plt_ctrl_nw is the asynchronous version of *plt_ctrl*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure is not used.

plt_ctrl and *plt_ctrl_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[XIO_DEVICE_FULL]	Another process is currently using the parallel port.

SEE ALSO

intro(3A), plt_data(3A).
xplot(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request a value of 0 is returned. Otherwise, a nonzero value is returned indicating one of two types of errors. If the request failed, one of the above failure codes is returned. If the output device did not accept the data, a negative status is returned and bits 0-2 reflect the state of the status signals from the device.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to nonzero if unsuccessful indicating one of two types of errors. If the request failed, one of the above failure codes is returned. If the output device did not accept the data, a negative status is returned and bits 0-2 reflect the state of the status signals from the device.

NAME

rplt_ctrl, rplt_ctrl_nw - send a control word to the ROP parallel port

SYNOPSIS

```
#include <sys/types.h>
#include <sys/immu.h>
#include <sys/pop.h>
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int rplt_ctrl (interface, status, ctrl, timeout, pulse)
int interface, status, ctrl, timeout, pulse;

int rplt_ctrl_nw (interface, status, ctrl, timeout, pulse, xiosb, efn)
int interface status, ctrl, timeout, pulse, efn;
struct xiosb *xiosb;
```

DESCRIPTION

rplt_ctrl provides a mechanism to send a single word through the control register on the parallel port residing on the Raster Operation Processor (ROP) graphics board. *Interface* is one of the following values defined in **<sys/pop.h>**: CENTRONICS, VERSATEC, or INTERGRAPH_DIFF. These values define the signal mapping for the target device. The user provides the three-bit mask, *status*, that determines which incoming signals should be considered when testing for the clear-to-send condition. Bits that are set in the mask indicate that the corresponding signals received from the device should be asserted. The format of the status word is as follows:

2	1	0
READY H	NO ERROR H	ONLINE H

Ctrl is a 16-bit word that will be sent to the device (bits 0-7 are data; bits 8-15 are control). The control bits are as follows:

15	14	13	12	11	10	9	8
NOT USED	RESET H	RFFED H	REOTR H	RLTER H	CLEAR H	PICLK H	PRINT H

Pulse indicates whether bit 8 (PRINT H) is toggled as a control bit or interpreted as a data bit.

Timeout indicates the time limit in 1/60 second intervals to wait for the device to become ready before aborting the request. This value is limited to 32767.

rplt_ctrl_nw is the asynchronous version of *rplt_ctrl*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure is not used.

rplt_ctrl and *rplt_ctrl_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[XIO_DEVICE_FULL]	Another process is currently using the parallel port.

SEE ALSO

intro(3A), plt_data(3A).
xplot(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, a nonzero value is returned indicating one of two types of errors. If the request failed, one of the above failure codes is returned. If the output device did not accept the data, a negative status is returned and bits 0-2 reflect the state of the status signals from the device.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to nonzero if unsuccessful indicating one of two types of errors. If the request failed, one of the above failure codes is returned. If the output device did not accept the data, a negative status is returned and bits 0-2 reflect the state of the status signals from the device.

NAME

plt_data, plt_data_nw - write data to the parallel port

SYNOPSIS

```
#include <sys/types.h>
#include <sys/immu.h>
#include <sys/pop.h>
#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>

int plt_data(interface, dbuf, dcnt, count, ctrl, timeout, nctrl,
                pulse, speed)
int interface, dcnt, count, ctrl, timeout, nctrl, pulse, speed;
char *dbuf;

int plt_data_nw(interface, dbuf, dcnt, count, ctrl, timeout, nctrl,
                pulse, speed, xiosb, efn)
int interface, dcnt, count, ctrl, timeout, nctrl, pulse, speed, efn;
char *dbuf;
struct xiosb *xiosb;
```

DESCRIPTION

plt_data writes data to a device through the parallel port. *Interface* is one of the following values defined in **<sys/pop.h>**: CENTRONICS, VERSATEC, or INTERGRAPH_DIFF. These values define how signals should be mapped for the target device. The only signal considered to determine a clear-to-send condition is the READY H signal.

Dbuf points to a the data buffer. *Dcnt* number of bytes are transferred from *dbuf* to the parallel port.

A mechanism is provided to accommodate devices that require a control signal at the end of each scan line. *Count* specifies the number of bytes to send from *dbuf* before sending *ctrl*. If the count is 0, no control word is sent. *Ctrl* is a 16-bit word that may be sent to the device (bits 0-7 are data; bits 8-15 are control). The control bits are as follows:

15	14	13	12	11	10	9	8
NOT USED	RESET H	RFFED H	REOTR H	RLTER H	CLEAR H	PICLK H	PRINT H

Nctrl is the control word that is sent with each byte in *dbuf*. The format of *nctrl* is the same as *ctrl* shown in the figure above. *Pulse* indicates whether bit 8 (PRINT H) is toggled as a control bit or interpreted as a data bit.

Timeout indicates the time limit in 1/60 second intervals to wait for the device to become ready before aborting the request. This value is limited to 32767.

plt_data_nw is the asynchronous version of *plt_data*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure is not used.

plt_data and *plt_data_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[BAD_DATA_BUFFER_ADDRESS]	The data buffer points to a invalid memory address.
[BAD_DATA_BUFFER_SIZE]	The size of the data buffer is invalid.
[PAGE_LOCK_FAILED]	The request is larger than the current available physical memory. The request should be broken up into smaller sizes.
[XIO_DEVICE_FULL]	The driver cannot accept any more requests until one completes. The current limit is 5.

SEE ALSO

intro(3A), plt_ctrl(3A).
xplot(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, a nonzero value is returned indicating one of two types of errors. If the request failed, one of the above failure codes is returned. If the output device did not accept the data, a negative status is returned and bits 0-2 reflect the state of the status signals from the device.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to nonzero if unsuccessful indicating one of two types of errors. If the request failed, one of the above failure codes is returned. If the output device did not accept the data, a negative status is returned and bits 0-2 reflect the state of the status signals from the device.

BUGS

The *speed* parameter is not currently implemented.

NAME

rplt_data, rplt_data_nw - write data to the ROP parallel port

SYNOPSIS

```
#include <sys/types.h>
#include <sys/immu.h>
#include <sys/pop.h>
#include <sys/xio/xio.h>
#include <sys/xio/xerr.h>

int rplt_data(interface, dbuf, dcnt, count, ctrl, timeout, nctrl,
              pulse, speed)
int interface, dcnt, count, ctrl, timeout, nctrl, pulse, speed;
char *dbuf;

int rplt_data_nw(interface, dbuf, dcnt, count, ctrl, timeout, nctrl,
                 pulse, speed, xiosb, efn)
int interface, dcnt, count, ctrl, timeout, nctrl, pulse, speed, efn;
char *dbuf;
struct xiosb *xiosb;
```

DESCRIPTION

rplt_data writes data to a device through the parallel port. *Interface* is one of the following values defined in **<sys/pop.h>**: CENTRONICS, VERSATEC, or INTERGRAPH_DIFF. These values define how signals should be mapped for the target device. The only signal considered to determine a clear-to-send condition is the READY H signal.

Dbuf points to a the data buffer. *Dcnt* number of bytes are transferred from *dbuf* to the parallel port.

A mechanism is provided to accommodate devices that require a control signal at the end of each scan line. *Count* specifies the number of bytes to send from *dbuf* before sending *ctrl*. If the count is 0, no control word is sent. *Ctrl* is a 16-bit word that may be sent to the device (bits 0-7 are data; bits 8-15 are control). The control bits are as follows:

15	14	13	12	11	10	9	8
NOT USED	RESET H	RFFED H	REOTR H	RLTER H	CLEAR H	PICLK H	PRINT H

Nctrl is the control word that is sent with each byte in *dbuf*. The format of *nctrl* is the same as *ctrl* shown in the figure above. *Pulse* indicates whether bit 8 (PRINT H) is toggled as a control bit or interpreted as a data bit.

Timeout indicates the time limit in 1/60 second intervals to wait for the device to become ready before aborting the request. This value is limited to 32767.

rplt_data_nw is the asynchronous version of *rplt_data*, providing the same capability without waiting for completion of the request. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated upon completion of the request (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure is not used.

rplt_data and *rplt_data_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[BAD_DATA_BUFFER_ADDRESS]	The data buffer points to a invalid memory address.
[BAD_DATA_BUFFER_SIZE]	The size of the data buffer is invalid.
[PAGE_LOCK_FAILED]	The request is larger than the current available physical memory. The request should be broken up into smaller sizes.
[XIO_DEVICE_FULL]	The driver cannot accept any more requests until one completes. The current limit is 5.

SEE ALSO

intro(3A), rplt_ctrl(3A).
xplot(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, a nonzero value is returned indicating one of two types of errors. If the request failed, one of the above failure codes is returned. If the output device did not accept the data, a negative status is returned and bits 0-2 reflect the state of the status signals from the device.

If the asynchronous request is accepted by the XIO system, a value of 0 will be returned by the request. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful, or to nonzero if unsuccessful indicating one of two types of errors. If the request failed, one of the above failure codes is returned. If the output device did not accept the data, a negative status is returned and bits 0-2 reflect the state of the status signals from the device.

BUGS

The *speed* parameter is not currently implemented.

NAME

rle_cancel - cancel outstanding asynchronous I/O on an RLE channel

SYNOPSIS

#include <sys/xio/xerr.h>

int rle_cancel (channel)
int channel;

DESCRIPTION

rle_cancel terminates all requests pending on the specified Run Length Encoding (RLE) *channel*. *Channel* must have been opened with *rle_open*(3A).

rle_cancel will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[RLE_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[RLE_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

rle_open(3A), rle_pipe_mem(3A).
xrle(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

rle_close - close an RLE channel

SYNOPSIS

```
#include <sys/xio/xerr.h>

int rle_close (channel)
int channel;
```

DESCRIPTION

rle_close terminates all requests pending on the specified Run Length Encoding (RLE) *channel* and closes the *channel*. *Channel* must have been opened with *rle_open*(3A).

rle_close will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[RLE_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[RLE_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

rle_open(3A).
xrle(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

rle_open - open an RLE channel

SYNOPSIS

#include <sys/xio/xerr.h>

int rle_open (channel)
int channel;

DESCRIPTION

rle_open opens the specified Run Length Encoding (RLE) *channel*. Only one process is allowed to open an RLE channel at a time. Each RLE board in the system is represented by a *channel* number so that *channel* 0 references the RLE board with the lowest Shared Resource (SR) Bus slot number.

rle_open will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[RLE_REDUNDANT_REQ]	The specified *channel* is currently opened by this process.
[RLE_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[RLE_CHANNEL_NOT_FOUND]	The specified *channel* is not in the system.
[RLE_CHANNEL_BUSY]	The specified *channel* is currently opened by another process.

SEE ALSO

xrle(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

rle_pipe_mem, rle_pipe_mem_nw - RLE from pipe to memory

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int rle_pipe_mem(channel, dbuf, dcnt, scanline, offset, timeout,
                 xfcnt)
int channel, dcnt, scanline, offset, timeout, *xfcnt;
char *dbuf;

int rle_pipe_mem_nw(channel, dbuf, dcnt, scanline, offset,
                    timeout, xiosb, efn)
int channel, dcnt, scanline, offset, timeout, efn;
char *dbuf;
struct xiosb *xiosb;
```

DESCRIPTION

rle_pipe_mem and *rle_pipe_mem_nw* provide a mechanism to run length encode data directly from the raster processing pipeline to virtual memory. The specified Run Length Encoding (RLE) *channel* will move the data. *Channel* must have been opened with *rle_open*(3A).

Dbuf points to the buffer where encoded data is to be written. The buffer must begin on a long word boundary. *Dcnt* indicates the buffer size in bytes and must be a multiple of four.

Scanline and *offset* indicate the initial values for the scanline and offset fields in the RLE header packets.

Timeout indicates the time limit in 1/60-second intervals for the data transfer. Any transfer that takes longer is aborted and an appropriate status is returned. A *timeout* value of zero disables the timeout function.

Upon completion of the synchronous request, *xfcnt* indicates the number of bytes transferred.

rle_pipe_mem_nw is the asynchronous version of *rle_pipe_mem*, providing the same capability without waiting the request to complete. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated when the request completes (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred.

rle_pipe_mem and *rle_pipe_mem_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[RLE_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.

[RLE_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[BAD_DATA_BUFFER_ADDRESS]	The data buffer either is not long word aligned or points to a nonwritable memory space.
[BAD_DATA_BUFFER_COUNT]	The byte count is not a multiple of four.
[PAGE_LOCK_FAILED]	Not enough physical memory for this request is available at this time.
[RLE_CANCELED]	The request was canceled with *rle_cancel*(3A) or *rle_close*(3A).
[RLE_COMP0_SYNC_ERROR]	A synchronization error occurred on component 0.
[RLE_COMP1_SYNC_ERROR]	A synchronization error occurred on component 1.
[RLE_COMP2_SYNC_ERROR]	A synchronization error occurred on component 2.
[RLE_OVERRUN_ERROR]	The output buffer is full and data remains to be processed.
[RLE_TIMEOUT]	The *timeout* expired before the transfer completed.

SEE ALSO

rle_open(3A), rle_close(3A), rle_cancel(3A).
intro(7A), xrle(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the XIO system accepts the asynchronous request, the request will return a 0. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to either 0 if successful or to one of the above failure codes if unsuccessful.

NAME

rle_setup - establish parameters for an RLE channel

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xrle.h>

int rle_setup (chan, linewidth, swathsize, top, left, igr, format)
int chan, linewidth, swathsize, top, left, igr, format;
```

DESCRIPTION

rle_setup establishes parameters for the specified Run Length Encoding (RLE) *chan*. *Chan* must have been opened with *rle_open*(3A). Possible values for *linewidth* are 32, 64, 128, 256, 512, and 1024. *Swathsize* represents the number of lines per swath. *Top* represents the size of the top margin to be skipped. *Left* represents the size of the left margin to be skipped.

Igr is the value to be placed in the IGR field of each scanline header; a value of -1 indicates that no scanline headers are to be produced.

Format determines which mode the RLE will operate in. Possible modes are defined in **<sys/xio/xrle.h>** and described below.

RLE_BW_RLE	black and white RLE
RLE_EKTRON_RLE	ektron format RLE
RLE_COLOR_RLE	color RLE
RLE_PASSTHRU1	single component; no compression
RLE_PASSTHRU3	all components; no compression
RLE_REVERSE_BW_RLE	reverse black and white RLE
RLE_REVERSE_EKTRON_RLE	reverse Ektron format RLE
RLE_REVERSE_COLOR_RLE	reverse color RLE
RLE_REVERSE_PASSTHRU1	reverse single component; no compression
RLE_REVERSE_PASSTHRU3	reverse all components; no compression

rle_setup will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[RLE_CHANNEL_INVALID]	The specified *chan* is beyond the maximum allowed.
[RLE_CHANNEL_NOT_OPEN]	The specified *chan* is not open for this process.
[RLE_INVALID_PARMS]	An invalid value was passed for a parameter.

SEE ALSO

rle_open(3A).
xrle(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

sif_cancel - cancel outstanding asynchronous I/O on a SIF channel

SYNOPSIS

#include <sys/xio/xerr.h>

int sif_cancel (channel)
int channel;

DESCRIPTION

sif_cancel terminates all requests pending on the specified Scanner Interface (SIF) *channel*. *Channel* must have been opened with *sif_open*(3A).

sif_cancel will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[SIF_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[SIF_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

sif_open(3A), sif_mem_pipe(3A), sif_scan_mem(3A), sif_scan_pipe(3A).
xsif(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

sif_close - close a SIF channel

SYNOPSIS

#include <sys/xio/xerr.h>

int sif_close (channel)
int channel;

DESCRIPTION

sif_close terminates all requests pending on the specified Scanner Interface (SIF) *channel* and closes it. *Channel* must have been opened with *sif_open*(3A).

sif_close will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[SIF_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[SIF_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.

SEE ALSO

sif_open(3A).
xsif(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

sif_open - open a SIF channel

SYNOPSIS

#include <sys/xio/xerr.h>

int sif_open (channel)
int channel;

DESCRIPTION

sif_open opens the specified Scanner Interface (SIF) *channel.* Only one process is allowed to open a SIF channel at a time. Each SIF board in the system is represented by a *channel* number so that *channel* 0 references the SIF board with the lowest Shared Resource (SR) Bus slot number.

sif_open will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[SIF_REDUNDANT_REQ]	The specified *channel* is currently open by this process.
[SIF_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[SIF_CHANNEL_NOT_FOUND]	The specified *channel* is not present in the system.
[SIF_CHANNEL_BUSY]	The specified *channel* is currently open by another process.

SEE ALSO

xsif(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

sif_scan_mem, sif_scan_mem_nw - transfer data from scanner to memory

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int sif_scan_mem (channel, dbuf, dcnt, timeout, xfcnt)
int channel, dcnt, timeout, *xfcnt;
char *dbuf;

int sif_scan_mem_nw (channel, dbuf, dcnt, timeout, xiosb, efn)
int channel, dcnt, timeout, efn;
char *dbuf;
struct xiosb *xiosb;
```

DESCRIPTION

sif_scan_mem and *sif_scan_mem_nw* provide a mechanism for transferring data directly from the scanner to virtual memory. The specified Scanner Interface (SIF) *channel* will move the data. *Channel* must have been opened with *sif_open*(3A).

Dbuf points to the buffer data is being transferred to. The buffer must begin on a long word boundary. *Dcnt* indicates the byte count to be transferred and must be a multiple of the *linewidth* specified in *sif_setup*(3A).

Timeout indicates the time limit in 1/60-second intervals for the data transfer. Any transfer that takes longer is aborted and an appropriate status is returned. A *timeout* value of zero disables the timeout function.

Upon completion of the synchronous request, *xfcnt* indicates the number of bytes transferred.

sif_scan_mem_nw is the asynchronous version of *sif_scan_mem*, providing the same capability without waiting the request to complete. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated when the request completes (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred.

sif_scan_mem and *sif_scan_mem_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[SIF_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[SIF_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[BAD_DATA_BUFFER_ADDRESS]	The data buffer is either not long word aligned or points to a nonwritable

	memory space.
[BAD_DATA_BUFFER_COUNT]	The byte count is not a multiple of *linewidth* from *sif_setup*(3A).
[PAGE_LOCK_FAILED]	Not enough physical memory for this request is available at this time.
[SIF_CANCELED]	The request was canceled with *sif_cancel*(3A) or *sif_close*(3A).
[SIF_PIX_PER_LINE_ERROR]	The scanner transmitted an incorrect number of pixels per scanline.
[SIF_LINE_PER_SWATH_ERROR]	The scanner transmitted an incorrect number of scanlines per swath.
[SIF_RED_PARITY_ERROR]	A parity error occurred on the red component input.
[SIF_GREEN_PARITY_ERROR]	A parity error occurred on the green component input.
[SIF_BLUE_PARITY_ERROR]	A parity error occurred on the blue component input.
[SIF_CYCLE_ERROR]	A hardware handshake error occurred on the transfer.
[SIF_TIMEOUT]	The *timeout* expired before the transfer completed.

SEE ALSO

intro(3A), sif_open(3A), sif_close(3A), sif_setup(3A), sif_cancel(3A).
xsif(7A) in the *CLIX System Administrator's Reference Manual.*

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the XIO system accepts the asynchronous request, the request will return a value of 0. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to 0 if successful, or one of the above failure codes if unsuccessful.

NAME

sif_scan_pipe, sif_scan_pipe_nw - transfer data from scanner to pipe

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xio.h>

int sif_scan_pipe (channel, dcnt, timeout, xfcnt)
int channel, dcnt, timeout, *xfcnt;

int sif_scan_pipe_nw (channel, dcnt, timeout, xiosb, efn)
int channel, dcnt, timeout, efn;
struct xiosb *xiosb;
```

DESCRIPTION

sif_scan_pipe and *sif_scan_pipe_nw* provide a mechanism for transferring data directly from the scanner to the scanner processing pipeline. The specified Scanner Interface (SIF) *channel* will move the data. *Channel* must have been opened with *sif_open*(3A).

Dcnt indicates the byte count to be transferred and must be a multiple of the *linewidth* specified in *sif_setup*(3A).

Timeout indicates the time limit in 1/60-second intervals for the data transfer. Any transfer that takes longer is aborted and an appropriate status is returned. A *timeout* value of zero disables the timeout function.

Upon completion of the synchronous request, *xfcnt* indicates the number of bytes transferred.

sif_scan_pipe_nw is the asynchronous version of *sif_scan_pipe*, providing the same capability without waiting the request to complete. *Efn* is the event flag number associated with the request. *Xiosb* is a pointer to the *xiosb* structure updated when the request completes (see *intro*(3A)). The *xfcnt* member of the *xiosb* structure indicates the number of bytes transferred.

sif_scan_pipe and *sif_scan_pipe_nw* will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request or *efn* is invalid.
[SIF_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[SIF_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[BAD_DATA_BUFFER_COUNT]	The byte count is not a multiple of *linewidth* from *sif_setup*(3A).
[SIF_CANCELED]	The request was canceled with *sif_cancel*(3A) or *sif_close*(3A).

[SIF_PIX_PER_LINE_ERROR]	The scanner transmitted an incorrect number of pixels per scanline.
[SIF_LINE_PER_SWATH_ERROR]	The scanner transmitted an incorrect number of scanlines per swath.
[SIF_RED_PARITY_ERROR]	A parity error occurred on the red component input.
[SIF_GREEN_PARITY_ERROR]	A parity error occurred on the green component input.
[SIF_BLUE_PARITY_ERROR]	A parity error occurred on the blue component input.
[SIF_CYCLE_ERROR]	A hardware handshake error occurred on the transfer.
[SIF_TIMEOUT]	The *timeout* expired before the transfer completed.

SEE ALSO

intro(3A), sif_open(3A), sif_close(3A), sif_setup(3A), sif_cancel(3A).
xsif(7A) in the *CLIX System Administrator's Reference Manual*.

DIAGNOSTICS

Upon successful completion of the synchronous request, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

If the XIO system accepts the asynchronous request, the request will return a value of 0. Otherwise, XIO_FAILURE will be returned. Upon completion of an accepted request, the *status* member of the *xiosb* structure will be set to 0 if successful, or one of the above failure codes if unsuccessful.

NAME

sif_setup - establish parameters for a SIF channel

SYNOPSIS

```
#include <sys/xio/xerr.h>
#include <sys/xio/xsif.h>

int sif_setup (channel, resolution, linewidth, swathsize, mode)
int channel, resolution, linewidth, swathsize, mode;
```

DESCRIPTION

sif_setup establishes Direct Memory Access (DMA) parameters for the specified Scanner Interface (SIF) *channel*. *Channel* must have been opened with *sif_open*(3A).

Resolution and *linewidth* are used together to determine the data hardware rejects automatically. This should reduce memory and time requirements for the system. The accepted data consists of the first *linewidth* bytes from each consecutive set of *resolution* bytes. All other data is rejected. If the concepts of *resolution* and *linewidth* do not apply, they should be set the same. Possible values for *linewidth* are 32, 64, 128, 256, 512, and 1024. Possible values for *resolution* are 256, 512, and 1024. *Linewidth* must always be less than or equal to *resolution*.

Swathsize represents the number of lines per swath.

Mode determines which color mode the SIF will operate in. Possible modes are defined in **<sys/xio/xsif.h>** and described below.

SIF_MONO_RED	Scan in monocolor mode using the red component.
SIF_MONO_GREEN	Scan in monocolor mode using the green component.
SIF_MONO_BLUE	Scan in monocolor mode using the blue component.
SIF_COLOR	Scan in color mode using all three components.

sif_setup will fail if one of the following is true:

[XIO_FAILURE]	The system does not contain the driver needed to support this request.
[SIF_CHANNEL_INVALID]	The specified *channel* is beyond the maximum allowed.
[SIF_CHANNEL_NOT_OPEN]	The specified *channel* is not open for this process.
[SIF_INVALID_PARMS]	The values for *resolution* and *linewidth* do not meet the stated requirements.

SEE ALSO

sif_open(3A).
xsif(7A) in the *CLIX System Administrator's Reference Manual*.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, one of the above failure codes is returned.

NAME

xio_allocef, xio_deallocef - allocate/deallocate an event flag number

SYNOPSIS

```
unsigned long xio_allocef (efn)
int *efn;

void xio_deallocef (mask)
unsigned long mask;
```

DESCRIPTION

xio_allocef allocates an event flag number used for asynchronous requests. *Efn* points to the integer location updated with an event flag number. The event flag number is an argument passed to an asynchronous routine to track the completion of the request. *xio_allocef* returns a mask that corresponds to the event flag number.

xio_deallocef deallocates event flag numbers obtained with *xio_allocef*. The bits set in *mask* correspond to the event flag numbers to be deallocated.

SEE ALSO

intro(3A), xio_readef(3A), xio_waitfr(3A), xio_notify(3A).

DIAGNOSTICS

Upon successful completion, of *xio_allocef*, a nonzero mask is returned. Otherwise, all event flag numbers have been allocated and a value of 0 is returned.

NAME

xio_notify - notify a process of an asynchronous request completion

SYNOPSIS

```
#include <sys/xio/xerr.h>

int xio_notify (mask, signal, var, val0, val1)
unsigned long mask;
int signal, var[2], val0, val1;
```

DESCRIPTION

xio_notify provides a mechanism to detect completion of an asynchronous request via signal or memory location update. Either or both of these methods of notification is available.

Mask corresponds to the event flag numbers of the asynchronous requests for which the process requires notification.

Signal is sent to the process upon completion of any of the specified asynchronous requests. The signal must be chosen in accordance with defined system signals, and the signal-catching handler must be established with standard signal management system calls (see *sigset*(2) or *signal*(2)).

Var is an array of two integers; *xio_notify* updates these locations with the values contained in *val0* and *val1*. *Var* must be aligned on a long-word boundary. If *var* is 0, this feature is disabled.

xio_notify will fail if either *mask* is 0, *signal* is not a valid signal number, or *var* is invalid.

SEE ALSO

intro(3A), xio_readef(3A), xio_waitfr(3A), xio_allocef(3A), signal(2), sigset(2).

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, XIO_FAILURE is returned.

NOTES

Only one *xio_notify* request (specifying either a signal notification, a memory location update, or both) may be active. If a subsequent *xio_notify* call is made, the previous call is no longer in effect.

WARNINGS

After notification, the process must call either *xio_readef*(3A), *xio_waitfr*(3A), *xio_wflor*(3A), or *xio_wfland*(3A) before the *xiosb* structures for the completed asynchronous requests are valid.

NAME

xio_readef, xio_clref, xio_setef - event flag mask functions

SYNOPSIS

unsigned long xio_readef ()

unsigned long xio_clref (mask)
unsigned long mask;

unsigned long xio_setef (mask)
unsigned long mask;

DESCRIPTION

xio_readef returns the event flag mask. Cleared bits in the mask possibly represent asynchronous requests that have not completed. The process is responsible for "knowing" which bits in the event flag mask are currently used by asynchronous requests.

xio_clref provides a mechanism to clear event flag numbers in the event flag mask. All bits set in *mask* will be cleared.

xio_setef provides a mechanism to set event flag numbers in the event flag mask. All bits set in *mask* will be set.

SEE ALSO

intro(3A), xio_allocef(3A), xio_waitfr(3A), xio_notify(3A).

DIAGNOSTICS

Upon successful completion, the event flag mask is returned. No errors are possible.

NAME

xio_waitfr, xio_wfland, xio_wflor - asynchronous event control

SYNOPSIS

unsigned long xio_waitfr (efn)
int efn;

unsigned long xio_wfland (mask)
unsigned long mask;

unsigned long xio_wflor (mask)
unsigned long mask;

DESCRIPTION

xio_waitfr provides a mechanism for a process to wait for the bit in the event flag mask corresponding to *efn* to be set. A bit set in the event flag mask usually corresponds to the completion of an asynchronous request. If no error occurred, the event flag mask is returned.

xio_wfland will return control to the caller when all bits in the event flag mask that correspond to all set bits in *mask* are set. This provides a mechanism for a process to wait for the completion of many outstanding asynchronous requests. The event flag mask is returned.

xio_wflor will return control to the caller when any of the bits in the event flag mask that correspond to set bits in *mask* are set. The call, in effect, waits for one of the outstanding asynchronous requests to complete. The event flag mask is returned.

SEE ALSO

intro(3A), xio_allocef(3A), xio_readef(3A), xio_notify(3A).

DIAGNOSTICS

Upon successful completion of *xio_waitfr*, the event flag mask is returned. Otherwise, a value of 0 is returned to indicate that *efn* is not valid. No errors are possible for *xio_wfland* and *xio_wflor*.

NAME

f77initio, f77uninitio - initialize or terminate FORTRAN I/O from C

SYNOPSIS

void f77initio ()
void f77uninitio ()

DESCRIPTION

f77initio initializes FORTRAN I/O by calling the appropriate routines in the FORTRAN library. *f77initio* must be called in programs that define their main routine in C and also use FORTRAN I/O. *f77uninitio* terminates FORTRAN I/O by calling the appropriate routines in the FORTRAN library. *f77uninitio* flushes data and closes any open FORTRAN logical unit numbers. These two routines are designed to be called from C source only.

SEE ALSO

fnum(3F), flush(3F), fdtounit(3F).

NAME

fdtounit - return FORTRAN logical unit associated with a file descriptor

SYNOPSIS

integer lun, fd

lun = fdtounit (fd)

DESCRIPTION

fdtounit returns the FORTRAN logical unit number associated with a particular file descriptor. *fdtounit* assumes the file descriptor has been returned by some past *open*(2) or *creat*(2) system call.

DIAGNOSTICS

Upon successful completion, the FORTRAN logical unit number is returned. Otherwise, a value of -1 is returned.

SEE ALSO

fnum(3F), f77initio(3F).
cexternal keyword in the *FORTRAN User's Guide*.
open(2), creat(2) in the *UNIX System V Programmer's Reference Manual.*

NAME

flush - flush the output for the specified FORTRAN logical unit

SYNOPSIS

integer i, lun

i = flush (lun)

DESCRIPTION

flush flushes the output for the specified FORTRAN logical unit. The FORTRAN library utilizes the buffered I/O routines *fopen*(3S), *fread*(3S), *fwrite*(3S), etc., for logical unit I/O. Thus, after a FORTRAN write statement is executed, some or all data may reside in a buffer and will not necessarily have been written to the device or file associated with the logical unit. Data will always be flushed when the program exits or a logical unit is closed. However, when a user requires data to be flushed to a device or file immediately, *flush* is provided.

DIAGNOSTICS

Upon successful completion, the FORTRAN logical unit number is returned. Otherwise, a value of -1 is returned.

SEE ALSO

fnum(3F), f77initio(3F).
fopen(3S), fclose(3S) in the *UNIX System V Programmer's Reference Manual.*

NAME

fnum - return the file descriptor associated with a FORTRAN logical unit

SYNOPSIS

integer i, lun

i = fnum (lun)

DESCRIPTION

fnum accepts a logical unit number as input and returns the associated file descriptor. The file descriptor is suitable for use in subsequent file-related system calls such as *read*(2), *write*(2), and *lseek*(2).

DIAGNOSTICS

Upon successful completion, the FORTRAN logical unit number is returned. Otherwise, a value of -1 is returned.

SEE ALSO

flush(3F), f77initio(3F).
cexternal keyword in the *FORTRAN User's Guide*.

File Formats (4)

NAME

intro - introduction to file formats

DESCRIPTION

This section outlines the formats of various files. The C structure declarations for the file formats are given where applicable. Usually, the header files containing these structure declarations can be found in the directory **/usr/include** or the directory **/usr/include/sys**. For inclusion in C language programs, however, the syntax **#include <***file-name***.h>** or **#include <sys/***file-name***.h>** should be used.

NAME

a.out - common assembler and link editor output

SYNOPSIS

#include <a.out.h>

DESCRIPTION

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

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

```
file header
CLIX system header
section 1 header
...
section n header
section 1 data
...
section n data
section 1 relocation
...
section n relocation
section 1 line numbers
...
section n line numbers
symbol table
string table
```

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

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

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

The *a.out* file produced by *ld*(1) has the magic number 0413 in the first field of the CLIX system header. The headers (file header, CLIX system header, and section headers) are loaded at the beginning of the text segment. The text immediately follows the headers in the user address space. The first text address will equal 0x0 plus the size of the headers, and will vary depending on the number of section headers in the *a.out* file. In an *a.out* file with three sections (.text, .data, and .bss), the first text address is at 0xd8 on the CLIPPER. The text segment is not writable by the program; if other processes are executing the same *a.out* file, the processes will share a text segment.

The data segment starts at the next 4M boundary past the last text address. The first data address is determined by the following. If an *a.out* file is split into 4K byte pages, one of the pages would contain both the end of text and the beginning of data. When the *core*(4) image is created, that page will appear twice: once at the end of text and once at the beginning of data (with some unused space in between). The duplicated page of text that appears at the beginning of data is never executed; it is duplicated so that the operating system may bring in pieces of the file in multiples of the page size without having to realign the beginning of the data section to a page boundary. Therefore, the first data address is the sum of the next segment boundary past the end of text plus the remainder of the last text address divided by 4K. If the last text address is a multiple of 4K, no duplication is necessary.

On the CLIPPER, the stack begins at location 0xC0000000 and grows toward lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the *brk*(2) system call.

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

File Header

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

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

CLIX System Header

The format of the CLIX system header is as follows:

```
typedef struct aouthdr
{
        short           magic;        /* magic number */
        short           vstamp;       /* version stamp */
        long            tsize;        /* text size in bytes, padded */
        long            dsize;        /* initialized data (.data) */
        long            bsize;        /* uninitialized data (.bss) */
        long            entry;        /* entry point */
        long            text_start;   /* base of text used for this file */
        long            data_start;   /* base of data used for this file */
        unsigned long   cliflags;     /* CLIPPER flags */
        unsigned char   tcache;       /* text region caching policy */
        unsigned char   dcache;       /* data region caching policy */
        unsigned char   scache;       /* stack region caching policy */
} AOUTHDR;
```

Section Header

The format of the section header is as follows:

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

Relocation

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

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

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

Symbol Table

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

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

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

#define n_name        _n._n_name
#define n_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 is as follows:

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

        struct {
                char    x_fname[FILNMLEN];
                } x_file;

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

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

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

SEE ALSO

as(1), cc(1), ld(1), core(4), reloc(4).
brk(2), filehdr(4), ldfcn(4), linenum(4), scnhdr(4), syms(4) in the *UNIX System V Programmer's Reference Manual.*

NAME

aliases - aliases file for *sendmail*(1M)

SYNOPSIS

/usr/lib/aliases

DESCRIPTION

The *aliases* file defines aliases used by *sendmail*(1M). Alias definitions in this file have one of the following formats:

> name: *name1* [, *name2*, ...]
> **name: :include:** *filename*
> **name:** "|*program*"

The first format above simply lists the addresses that should be aliased to the alias name. The second format specifies a file that contains addresses listed one per line. For example, the following alias would cause *sendmail*(1M) to read the "/usr/local/poets.list" file for a list of recipients:

```
poets: :include:/usr/local/poets.list
```

The third format specifies a program to which mail messages should be piped. The double quotation marks are necessary to prevent *sendmail*(1M) from suppressing the blanks between arguments. For example, the following alias would cause *sendmail*(1M) to pipe mail messages to **stdin** of the program called "/usr/frank/mymailer":

```
mymailer: "|/usr/frank/mymailer -a"
```

Complete path names must be furnished in the second and third formats.

Only local names may be aliased. In other words, an alias name cannot contain a **!** or **@** character. For example, the following would not have the desired effect:

```
eric@mit-xx: eric@berkeley.EDU
```

Aliases may be continued by starting any continuation lines with a space or tab. Blank lines and lines beginning with a pound sign (#) are treated as comments.

When *sendmail*(1M) is first installed on a host and when the *aliases* file is modified, *newaliases*(1) should be invoked to rebuild the database and create the *ndbm*(3B) files **/usr/lib/aliases.dir** and **/usr/lib/aliases.pag**. *sendmail*(1M) reads these files to resolve aliases. Reading these files instead of the *aliases* file itself improves performance.

If the contents of the file in the second format or the program in the third format have been modified, *newaliases*(1) does not need to be invoked.

After aliasing is performed, local and valid recipients who have a **.forward** file in their home directory have messages forwarded to the list of users defined in that file.

SEE ALSO

newaliases(1), ndbm(3B).
sendmail(1M) in the *CLIX System Administrator's Reference Manual.*

CAVEATS

Because of restrictions in *ndbm*(3B), a single alias cannot contain more than 1000 bytes of information. Longer aliases may by implemented by chaining. Chaining involves making the last name in the alias a dummy name that is a continuation alias.

NAME

ansitape - ANSI standard magtape labels

DESCRIPTION

An ANSI-labeled tape starts with a volume header. This header specifies the volume name and protection, the owner of the volume, and the ANSI label standard level that the tape conforms to.

Every file on the tape has a header, data blocks, and a trailer. A tape mark follows each element. At the end of the tape, two tape marks follow the trailer to indicate logical end-of-tape.

If a file is too large to be copied on one tape, it may be continued on another tape by modifying the trailer section.

VOLUME HEADER			
Field	Width	Example	Use
VOL1	4	VOL1	Indicates this is a volume header.
Label	6	VAX1	The name of the volume.
Access	1	<SPACE>	Volume protection. <SPACE> means unprotected.
IGN1	20		<< ignored >>
IGN2	6		<< ignored >>
Owner	14	Joe User	The name of the user.
IGN3	28		<< ignored >>
Level	1	3	ANSI standard level.

Owner	The owner field is 14 characters in ANSI labels. IBM labels cut the owner field to 10 characters. The IGN2 field is 10 characters on IBM-format tapes.

FILE HEADER 1			
Field	Width	Example	Use
HDR1	4	HDR1	Identifies first file header.
Name	17	FILE.DAT	Leftmost 17 characters of file name.
Set Name	6	VAX1	Name of volume set this file is part of.
Vol Num	4	0001	Number of this volume within volume set.
File Num	4	0001	Number of file on this tape.
Generation	4	0001	Resembles a major release number.
Gen Version	2	00	Version of a file within a release.
Created	6	b86001	The date of file creation.
Expires	6	b86365	Date file expires.
Access	1	<SPACE>	File protection. <SPACE> means unprotected.
Blockcount	6	000000	Number of blocks in the file.
System	13	OS360	The name of the software system that created the tape.
IGN	7		<< ignored >>

Name	The file name may have up to 17 characters in IBM labels, and ANSI labels before standard level 3. On ANSI level 3 and after, the HDR4 record provides overflow storage for up to 63 more characters of the file name.
Set Name	On multireel tape sets, a name identifying the set as a whole. Normally, this is just the volume name of the first reel in the set.
Generation	Resembles a major release number. The version field is a version within a generation. On VAX/VMS systems, these two fields are mathematically related to the (single) version number of disk files.
Created	The date the file was created. This is a six-character field; the first character is always a <SPACE>. The next two are the year. The final three are the day within the year, counting January 1st as day 1.
Blocks	The number of blocks in the file. In HDR1 records, this is always 0. The corresponding EOF1 or EOV1 contains the number of tape blocks written in the file on the current reel.

FILE HEADER 2			
Field	Width	Example	Use
HDR2	4	HDR2	Second file header.
Rec Format	1	D	Record format.
Blk Length	5	02048	Tape block size.
Rec Length	5	00080	Record size.
Density	1	3	Recording density code.
Vol Switch	1	0	1 if this is a continuation of a file from a previous reel.
Job	17	user/program	See following notes.
Recording	2	<SPACE>	Unused in 9-track tapes.
Car Control	1	<SPACE>	See following notes.
Blocking	1	B	See following notes.
IGN	11		<< ignored >>
Offset	2	00	Bytes to skip at front of each block.

Rec Format A single character indicating what type of records are provided. The codes are as follows:

Code	Meaning
F	Fixed-length
D	Variable up to rec length
V	IBM code for variable
U	Unknown

Job The name of the job (user name in CLIX) right-padded to eight characters, a slash (/), and the job step (program name in CLIX) right-padded to eight characters. This identifies where the JCL was located when this file was created.

Carriage Control
Normally a <SPACE>, indicating that the records do not contain carriage control information. When printed, each record is placed on a separate line. If an "A" is used, the first character of each record is presumed to be a FORTRAN carriage-control character. VAX/VMS also uses "M" to indicate that carriage-control is embedded as part of the data. This is usually used in the case of binary files.

Blocking The B indicates that the number of records that will fit are placed in a physical tape block. Records do not cross block boundaries. A <SPACE> indicates only one record per physical tape block.

The HDR3 and HDR4 labels are not written on IBM tapes. ANSI allows but does not require these labels.

FILE HEADER 3			
Field	Width	Example	Use
HDR3	4	HDR3	Third file header.
OS	76		Operating-system dependent.

OS This field is reserved for the operating system that created the file to use. Other operating systems should disregard HDR3 records. On VAX/VMS, this record contains the RMS file description.

FILE HEADER 4			
Field	Width	Example	Use
HDR4	4	HDR4	Fourth file header.
Name 2	63		Name continuation from HDR1.
Unknown	2	00	Unknown, fill with 00.
IGN	11		<< ignored >>

Name 2
On ANSI tapes, if the file name is longer than 17 characters, the first 17 are placed in the HDR1 record. The next 63 are put in HDR4. File names longer than 80 characters are truncated. A HDR3 record is not required in order to have a HDR4.

File Trailing Labels

These labels are written after a tape file. Every label written at the head of the file will have a corresponding label at the tail. Except for the *block count* field in HDR1, the only difference is in the name of the label. If we have reached the logical end of the file, the characters HDR in the headers are replaced by the characters EOF in the trailing labels. If we are not at the logical end of the file, but are merely pausing at the physical end of tape before continuing on another reel, the HDR characters are replaced by EOV (end-of-volume).

The *blockcount* field of HDR1 was initially recorded as 000000. When the trailers are written, the block count is changed to indicate the number of tape data blocks written. A file continued over several volumes maintains separate counts for each reel.

Record Formats

The two basic record formats are fixed and variable.

Fixed format uses records that all have a constant length. This is true with VAX/VMS executable images (record length = 512). It is also used by IBM systems for text files, with a record length of 80 (card images). The record size field of HDR2 tells the length of each record.

With fixed-length records, the *block-size* is usually selected to be some multiple of the *record-size*. The number of records that will fit is placed in each block. Since records do not (normally) span physical tape blocks, extra space at the end of a block is wasted.

Variable-length records are used by VAX/VMS for text files. The CLIX program *ansitape*(1) also turns CLIX text files into variable-length tape files. With this format, the record length specified in HDR2 is an upper limit.

Each record is proceeded by a four digit (zero-filled) byte count. The count included the digits themselves, so the minimum valid number is 0004. These four digits specify the length of the record. The data follows the digits, and is, in turn, followed by the digits for the next record.

When writing, *ansitape*(1) checks to ensure enough room is in the tape block for the next record. If the record (including its length digits) will not fit, the current block is sent to the tape and a new block is started. Unused space at the end of the tape block is filled with circumflex (^) characters.

SEE ALSO

ansitape(1).

NAME

backup, dumpdates - incremental dump format

SYNOPSIS

#include <sys/types.h>
#include <sys/inode.h>
#include <protocols/dumprestore.h>

DESCRIPTION

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

A header record
Two groups of bit map records
A group of records describing directories
A group of records describing files

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

```
#define TP_BSIZE       1024
#define NTREC          10
#define TP_NINDIR      (TP_BSIZE/2)

#define TS_TAPE        1
#define TS_INODE       2
#define TS_BITS        3
#define TS_ADDR        4
#define TS_END         5
#define TS_CLRI        6
#define OFS_MAGIC      (int) 60011
#define NFS_MAGIC      (int) 60012
#define CHECKSUM       (int) 84446

struct u_spcl {
        char dummy[TP_BSIZE];
        struct  s_spcl {
                int         c_type;
                time_t      c_date;
                time_t      c_ddate;
                int         c_volume;
                daddr_t     c_tapea;
                ino_t       c_inumber;
                int         c_magic;
                int         c_checksum;
#ifdef FFS
                struct      ffsdinode  c_dinode;
#else
                struct      dinode     c_dinode;
#endif
                int         c_count;
                char        c_addr[TP_NINDIR];
                } s_spcl;
        } u_spcl;
        #define spcl u_spcl.s_spcl
        #defineDUMPOUTFMT "%-20s %c %s" /* for printf */
                                         /* name, incno, ctime(date) */
        #defineDUMPINFMT  "%20s %c %[^\n]\n"   /* inverse for scanf */
```

NTREC is the number of 1024-byte records in a physical tape block. TP_BSIZE is the size of file blocks on the dump tapes. NTREC is the number of TP_BSIZE blocks written in each tape record. TP_NINDIR is the number of indirect pointers in a TS_ADDR record. It must be a power of two.

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

TS_TAPE	Tape volume label.
TS_INODE	A file or directory follows. The *c_dinode* field is a copy of the disk i-node and contains bits telling the type of file this is.
TS_BITS	A bit map follows. This bit map has a 1 bit for each i-node that was backed up.
TS_ADDR	A subrecord of a file description. See *c_addr* below.
TS_END	End of tape record.
TS_CLRI	A bit map follows. This bit map contains a zero bit for all i-nodes that were empty on the file system when backed up.
MAGIC	All header records have this number in *c_magic*. OFS_MAGIC is the old file system magic number. NFS_MAGIC is the new file system magic number.
CHECKSUM	Header records checksum to this value.

The fields of the header structure are as follows:

c_type	The type of the header.
c_date	The date the backup was taken.
c_ddate	The date the file system was backed up from.
c_volume	The current volume number of the backup.
c_tapea	The current number of this (1024-byte) record.
c_inumber	The number of the i-node being backed up if this has type TS_INODE.
c_magic	The value NFS_MAGIC above, truncated as needed.
c_checksum	The value needed to make the record sum to CHECKSUM.
c_dinode	A copy of the i-node as it appears on the file system.

c_count The count of characters in *c_addr*.

c_addr An array of characters describing the blocks of the backed up file. A character is zero if the block associated with that character was not present on the file system. Otherwise, the character is nonzero. If the block was not on the file system, no block was backed up; the block will be restored as a hole in the file. If space in this record is not sufficient to describe all blocks in a file, TS_ADDR records will be scattered through the file, each picking up where the last left off.

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

The structure *idates* describes an entry in the file **/etc/dumpdates** where backup history is kept.

```
struct idates {
        char      id_name[16];
        char      id_incno;
        time_t    id_ddate;
};
```

The fields of the structure are as follows:

id_name The dumped file system is **/dev/dsk/s?u?p?.?**.

id_incno The level number of the dump tape; see *backup*(1).

id_ddate The date of the incremental backup.

FILES

/etc/dumpdates

SEE ALSO

backup(1), restore(1).

NAME

bootheader - boot file header format

DESCRIPTION

System processors have individual boot files containing a *bootheader*, executable code, and initialized data. The *bootheader* starts at the beginning of the first block of the file, and the processor specific code and data start at the beginning of the second block of the file.

The format of the *bootheader* is as follows:

```
struct bootheader {
        long      b_magic;       /* magic number = 0x5441534b */
        u_short   b_checksum;    /* file checksum */
        short     b_processor;   /* processor type */
        long      b_loadaddr;    /* load address */
        u_long    b_loadsize;    /* code and initialized data size */
        long      b_uinitaddr;   /* uninitialized data start address */
        u_long    b_uinitsize;   /* uninitialized data size */
        long      b_entry;       /* start address */
        long      b_time;        /* time of file creation */
};
```

The members of this structure are as follows:

b_magic A 32-bit signed integer value that identifies the boot file type.

b_checksum The two's complement of the 16-bit unsigned sum of the boot file including the boot header block.

b_processor The type of processor to be loaded using the information in the *bootheader*. The valid types are listed below:

```
#define PROC_IOP     0    /* IOP */
#define PROC_UE      1    /* CLIPPER engine */
#define PROC_ROP     2    /* ROP */
#define PROC_DATA    3    /* raw data */
#define PROC_DIAG    4    /* diagnostics */
#define PROC_IFB     5    /* IFB */
#define PROC_FPE     6    /* FPE */
#define PROC_DIG     7    /* digitizer */
```

b_loadaddr The address where processor code and initialized data are loaded.

b_loadsize A 32-bit unsigned integer that specifies the number of bytes to be loaded.

b_uinitaddr The address where uninitialized data is loaded.

b_uinitsize A 32-bit unsigned integer that specifies the number of bytes to be filled.

b_entry	The location where the booted processor begins execution.
b_time	Encoded in seconds since 00:00:00 GMT, January 1, 1970.

SEE ALSO

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

CAVEATS

Some processors can only begin execution at a fixed address. Therefore, the *b_entry* field on these processors will be ignored.

NAME

certnote.com - Intergraph software certification documentation file

DESCRIPTION

certnote.com is a documentation file delivered with every Intergraph software product. *certnote.com* documents Trouble Reports (TRs) and Change Requests (CRs) filed on a product at the time of its release. Also noted are potential pitfalls, troubleshooting hints, and suggested ways to avoid problems with the product.

A *certnote.com* file consists of a header section followed by the certification comments. The following is an example header section:

```
**************************************************************************
Product:            System V 3.1 Boot Images
Product version:    5.3.1
Product No.:        SS043       Fixes date: 23-AUG-1988
Operating System:   System V
Menus:              N/A

Hardware Dependencies:
    InterPro® 120/InterView® 120
    InterPro 32C/InterAct® 32C/InterView 32C
    InterServe™ 200
    InterPro 220/InterAct 220/InterView 220
    InterPro 240/InterPro 245
    InterServe 300/InterServe 305/InterServe 400
    InterPro 340/InterAct 340/InterView 340
    InterPro 360/InterAct 360/InterView 360

Software Dependencies:
    None.

Document Name:
    None.
**************************************************************************
```

The first section of the header identifies the product. Where applicable, the "Menus" field identifies the Intergraph paper menu associated with the product.

"Hardware Dependencies" lists the systems on which the product runs.

"Software Dependencies" lists other software products necessary for proper use of this product.

"Document Name" lists the documents and manuals available for purchase or included with purchase of the product.

The following is an example comment section:

```
Comments:

  o First 4000 (or 8000 depending on disk type) blocks of the
    hard disk are allocated for boot images.  At download, at
    least 4000 blocks must be available for temporary
    (download only) space.

Conditions:

  o This product must be loaded by the "newprod" utility
    located in the DELTOOLS product.

The following trouble reports are currently open against this
product:

  o None.

The following change requests are currently open against this
product:

  CR# 88I0000
    Would like to be able to change the peripheral config-
    uration, especially floating menu present, without having
    to reboot.
```

The comment section is designed to communicate documented problems with the product to the user.

SEE ALSO

fixes.com(4).
newprod(1M) in the *CLIX System Administrator's Reference Manual.*

NOTES

certnote.com files generally reside on Intergraph delivery systems. They are downloaded to Intergraph CLIX-based systems during product installation for local examination.

NAME

clh - Intergraph network clearinghouse database

DESCRIPTION

The Intergraph clearinghouse is a network-wide distributed database. Its main use is to bind names to network addresses so that users do not need to use addresses. It can also be used to access any information that needs to be available to the network.

The clearinghouse **nodes** directory has three subdirectories: **local**, **owned**, and **heard**. These subdirectories store information about the nodes and individuals on the network. The files in these subdirectories are called *objects*. Each clearinghouse *object* is a text file stored in at least one of the subdirectories. An *object* name can have up to 14 characters and contain only lowercase alphanumeric characters and underscores. In each *object*'s file is a list of one or more properties and property values with the form:

property:*value*

Although an entry can contain any property name, the following "well-know" properties are used by programs that access the clearinghouse:

address	A network address stored as a hexadecimal character string. (The maximum size is 26 characters.)
node-name	An *object* representing a node that the clearinghouse can use to look up a network address (maximum size is 14 characters).
alias	Same as **node-name**.
scope	A list of Local Area Networks (LANs) to which this *object* should be propagated. (The *object* is always propagated to the local LAN.)
namex	*namex*(1) uses this property to indicate that it created or modified the *object*.
tcp_address	A TCP/IP Internet address stored as a decimal, hexadecimal, or octal character string. (The maximum size is 19 characters.)
netmap_info	A formatted line containing information about the system and its components. This line is added or updated automatically at system startup.
owner	The hexadecimal network address of the node from which the *object* was broadcast. This property is added automatically when the *object* is received from the network and therefore should only appear in *objects* in the **heard** subdirectory.

The following entries are provided for network maintenance purposes:

primary_user	The name of the primary user of the local node.

phone	The phone number of the **primary_user**.
location	The location of the local node.

EXAMPLES

```
address:        000b2345.08-00-36-76-09-00
tcp_address:    122.9.100.6
scope:  000b2343,b2344
scope:  b2343
NETMAP_info   : UNIX System V rel 3.1,
Primary_user  : John Smith
Location: Bismarck, North Dakota
Phone  :701-555-1212
Favorite color  : green
```

FILES

/usr/lib/nodes/local
/usr/lib/nodes/heard
/usr/lib/nodes/owned

SEE ALSO

clh(1).
"Network Programming Guide" in the *CLIX System Guide*.

NAME

core - format of core image file

DESCRIPTION

The CLIX system writes out a *core* image of a terminated process when certain errors occur. See *signal*(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The *core* image is called **core** and is written in the process's working directory (if allowed, normal access controls apply). A process with an effective user ID different from the real user ID will not produce a *core* image.

The first section of the *core* image is a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. The size of this section depends on the parameter *usize*, which is defined in **<sys/param.h>**. The remainder represents the actual contents (excluding the text area) of the user's *core* area when the *core* image was written. For a paged image file, which is standard, the dump of the data segment begins after any portion overlapped with the text section. Each segment in the *core* file begins on a page boundary.

The format of the information in the first section is described by the *user* structure of the system, defined in **<sys/user.h>**. Not included in this file are the locations of the registers. These are outlined in **<sys/reg.h>**.

SEE ALSO

adb(1), sdb(1), signal(2).
nocore(1M) in the *CLIX System Administrator's Reference Manual.*
setuid(2) in the *UNIX System V Programmer's Reference Manual.*

NAME

diskpar - disk partition header format

DESCRIPTION

All Intergraph disks are divided into sections called partitions. Each disk partition is preceded by a single-block *diskpar* header which provides information about the partition. The partition number, the modifier number, the partition size in blocks, and the disk partition magic number are contained in the *diskpar* header. The organization of the *diskpar* header is listed below.

Byte Offset	Description
0	partition number
1	modifier number
2-5	size in blocks
6-9	magic number

The partition size and the magic number are 32-bit unsigned integer values with the least significant byte of each value corresponding to the least significant byte of its header location.

The disk partition magic number is 0x454e4153.

SEE ALSO

mkpar(1M), parck(1M), dc(7S) in the *CLIX System Administrator's Reference Manual.*

NAME

disktab - disk description file

DESCRIPTION

disktab is a simple database which describes disk geometries and disk partition characteristics. Entries in *disktab* consist of a number of ":" separated fields. The first entry for each disk gives the names which are known for the disk, separated by "|" characters. The last name given should be a long name which uniquely identifies the disk.

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

Name	Type	Description
ns	num	number of sectors per track
nt	num	number of tracks per cylinder
nc	num	total number of cylinders on the disk
se	num	sector size in bytes
sf	bool	supports bad144-style bad sector forwarding
so	bool	partition offsets in sectors
ty	str	Type of disk (e.g., removable or winchester)

The following are for 4.3 Berkeley Software Distribution (BSD) compatibility:

Name	Type	Description
ba	num	block size for partition (bytes)
fa	num	fragment size for partition (bytes)
pa	num	size of partition in sectors

FILES

/etc/disktab

SEE ALSO

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

NAME

errord.rc - error log configuration file

SYNOPSIS

/usr/adm/errord.rc

DESCRIPTION

errord.rc contains configuration information for the error daemon, *errord*(1M). The set of keywords understood by *errord*(1M) are as follows:

function= Send error messages to a remote system if **s** is specified, receive error messages from remote systems if **r** is specified, and do not send or receive error messages from remote systems if **n** is specified. **n** is the default.

node= Specify the node name of the system to send errors to.

name= Specify the name of the file to be used as the error log file. The default is **/usr/adm/errlog**.

max= Specify the maximum size (in blocks) of the error log file. When the error log file reaches this size, it is renamed **/usr/adm/olderrorlog** and a new error log file is created. The default maximum size is 500 blocks.

include= Specify a list of error types to log in the error log file. If an **include=** line exists and an error type is not listed on the **include=** line, that error type is not logged. If multiple **include=** lines appear in the file, only the error types specified on the first **include=** line will be logged. The default action of *errord*(1M) is to log all error types.

Valid error types for **include=** are as follows: **device**, **user**, **panic**, **memory**, **slave**, **disk**, **tape**, **floppy**, **asycn**, **scan**, **parallel**, **digitizer**, **timeout**, **security**, **stray**, **optic**, **soft**, **retry** and **hard**.

exclude= Specify a list of error types not to log in the error log file. If an error type is not listed on an **exclude=** line, that error type is logged. If multiple **exclude=** lines appear in the file, only the error types specified on the first **exclude=** line will not be logged. The default action of the daemon is to log all error types. Valid error types for **exclude=** are the same as the error types for **include=**.

Comment. All text following the **#** character will be ignored by *errord*(1M).

EXAMPLES

```
#Send errors to node bike.  Do not log retries, soft errors, or tape errors.
function=s
node=bike
#Notice there is no space around the =.
#Log local errors to /usr/tmp/err.
name=/usr/tmp/err
exclude=retry,soft,tape
```

FILES

/usr/adm/errlog	default system error log file
/usr/adm/olderrlog	old system error log file
/usr/adm/errord.rc	error daemon configuration file

SEE ALSO

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

NAME

exports - NFS file systems being exported

DESCRIPTION

The file **/etc/exports** describes the file systems which are being exported to Network File System (NFS) clients. It is created by the system administrator using a text editor and processed by the mount request daemon *mountd*(1M) each time a mount request is received.

The file consists of a list of file systems and the *netgroups*(4) or machine names that can remotely mount each file system. The file system names are left justified and followed by a list of names separated by white space. The names are searched for first in **/etc/netgroups** and then in **/etc/hosts**. A file system name with no name list means "export to world". A "#" anywhere in the file indicates a comment extending to the end of the line it appears on. Lines beginning with white space are continuation lines.

EXAMPLES

```
/usr          clients          # export to my clients
/usr/local                     # export to the world
/usr2         ip ingr opus     # export to only these machines
```

FILES

/etc/exports

SEE ALSO

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

BUGS

The identification of the remote system is dependent on the local network transport mechanism employed.

NAME

ffsfs - format of file system volume

SYNOPSIS

#include <sys/types.h>
#include <sys/fs/ffsfs.h>
#include <sys/fs/ffsinode.h>

DESCRIPTION

Every file system storage volume (disk or nine-track tape, for instance) has a common format for certain vital information. Every such volume is divided into a certain number of blocks. The block size is a parameter of the file system.

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

```
#define         FS_MAGIC    0x011954

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

```
        long       fs_fragshift;          /* block to frag shift */
        long       fs_fsbtodb;            /* fsbtodb and dbtofsb shift constant */
        long       fs_sbsize;             /* actual size of super block */
        long       fs_csmask;             /* csum block offset */
        long       fs_csshift;            /* csum block number */
        long       fs_nindir;             /* value of NINDIR */
        long       fs_inopb;              /* value of INOPB */
        long       fs_nspf;               /* value of NSPF */
        long       fs_optim;              /* optimization preference, see below */
        long       fs_sparecon[5];        /* reserved for future constants */
    /* sizes determined by number of cylinder groups and their sizes */
        daddr_t    fs_csaddr;             /* blk addr of cyl grp summary area */
        long       fs_cssize;             /* size of cyl grp summary area */
        long       fs_cgsize;             /* cylinder group size */
    /* these fields should be derived from the hardware */
        long       fs_ntrak;              /* tracks per cylinder */
        long       fs_nsect;              /* sectors per track */
        long       fs_spc;                /* sectors per cylinder */
    /* this comes from the disk driver partitioning */
        long       fs_ncyl;               /* cylinders in file system */
    /* these fields can be computed from the others */
        long       fs_cpg;                /* cylinders per group */
        long       fs_ipg;                /* inodes per group */
        long       fs_fpg;                /* blocks per group * fs_frag */
    /* this data must be re-computed after crashes */
        struct     csum fs_cstotal;       /* cylinder summary information */
    /* these fields are cleared at mount time */
        long       fs_clean;              /* file system is clean flag */
        char       fs_fmod;               /* super block modified flag */
        char       fs_clean;              /* unused byte */
        char       fs_ronly;              /* mounted read-only flag */
        char       fs_flags;              /* currently unused flag */
        char       fs_fsmnt[MAXMNTLEN]; /* name mounted on */
    /* these fields retain the current block allocation info */
        long       fs_cgrotor;            /* last cg searched */
        struct     csum *fs_csp[MAXCSBUFS]; /* list of fs_cs info buffers */
        long       fs_cpc;                /* cyl per cycle in postbl */
        short      fs_postbl[MAXCPG][NRPOS]; /* head of blocks for each rotation */
        long       fs_magic;              /* magic number */
        char       fs_fname[6];           /* used by labelit */
        char       fs_fpack[6];           /* used by labelit */
        u_char     fs_rotbl[1];           /* list of blocks for each rotation */
    /* actually longer */
    };
```

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

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

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

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

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

The root i-node is the root of the file system. I-node 0 can't be used for normal purposes and historically bad blocks were linked to i-node 1, thus the root i-node is 2 (i-node 1 is no longer used for this purpose, however numerous dump tapes make this assumption, so we are stuck with it). The **lost+found** directory is given the next available i-node when it is initially created by *ffsmkfs*(1M).

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

Empirically the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 4, thus the default fragment size is a fourth of the block size.

fs_optim specifies whether the file system should try to minimize the time spent allocating blocks, or if it should attempt to minimize the space fragmentation on the disk. If the value of *fs_minfree* (see above) is less than 10%, then the file system defaults to optimizing for space to avoid running out of full sized blocks. If the value of minfree is greater than or equal to 10%, fragmentation is unlikely to be problematical, and the file system defaults to optimizing for time.

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

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

Each file system has a statically allocated number of i-nodes. An i-node is allocated for each NBPI bytes of disk space. The i-node allocation strategy is extremely conservative.

MAXIPG bounds the number of i-nodes per cylinder group, and is needed only to keep the structure simpler by having the only a single variable size element (the free bit map). MAXIPG must be a multiple of INOPB(fs).

MINBSIZE is the smallest allowable block size. With a MINBSIZE of 4096 it is possible to create files of size 2^32 with only two levels of indirection. MINBSIZE must be big enough to hold a cylinder group block, thus changes to the *cg* structure must keep its size within MINBSIZE. MAXCPG is limited only to dimension an array in the *cg* structure; it can be made larger as long as that structure's size remains within the bounds dictated by MINBSIZE. Note that super-blocks are never more than size SBSIZE.

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

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

The size of the *csum* structure must be a power of two in order for the *fs_cs* macro to work.

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

MAXBPG bounds the number of blocks of data per cylinder group, and is limited by the fact that cylinder groups are at most one block. The size of the free block table is derived from the size of blocks and the number of remaining bytes in the cylinder group structure the *cg* structure.

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

NAME

ffsinode - structure of an FFS disk i-node

SYNOPSIS

#include <sys/types.h>
#include <sys/fs/ffsinode.h>

DESCRIPTION

An i-node for a plain file or directory has the following structure defined in **<sys/fs/ffsinode.h>**.

```
struct icommon {
        u_short    ic_mode;            /*  0: mode and type of file */
        short      ic_nlink;           /*  2: number of links to file */
        uid_t      ic_uid;             /*  4: owner's user ID */
        gid_t      ic_gid;             /*  6: owner's group ID */
        quad       ic_size;            /*  8: number of bytes in file */
        time_t     ic_atime;           /* 16: time last accessed */
        long       ic_atspare;
        time_t     ic_mtime;           /* 24: time last modified */
        long       ic_mtspare;
        time_t     ic_ctime;           /* 32: last time inode changed */
        long       ic_ctspare;
        daddr_t    ic_db[NDADDR];      /* 40: disk block addresses */
        daddr_t    ic_ib[NIADDR];      /* 88: indirect blocks */
        long       ic_flags;           /* 100: status, currently unused */
        long       ic_blocks;          /* 104: blocks actually held */
        long       ic_gen;             /* 108: generation count for NFS */
        long       ic_spare[4];        /* 112: reserved, currently unused */
};

struct ffsdinode {
        union {
                struct    icommon di_icom;
                char      di_size[128];
        } di_un;
};
```

SEE ALSO

ffsfs(4).
stat(2) in the *UNIX System V Programmer's Reference Manual.*

NAME

fixes.com - Intergraph software delivery documentation file

DESCRIPTION

fixes.com is a documentation file delivered with every Intergraph software product. A *fixes.com* file consists of a header section followed by one or more delivery, or "fix", sections.

The following is an example header section:

```
;       PRODUCT NUMBER          : SS043
;       PRODUCT NAME            : System V 3.1 Boot Images
;       PRODUCT DIRECTORY       : UNIXBOOT
;       PRODUCT VERSION         : 5.3.1
```

In the header illustrated above, the colon is a delimiter and the names to the left of the colon are keywords. The important fields are the product number and the product version number. The five character product number is a unique Intergraph-assigned identifier used by some Intergraph software installation utilities. The product version number is also used by Intergraph software installation utilities to distinguish among major software releases. The product name and product directory are informational fields.

The following is an example "fix" section:

```
;       DATE                    : 23-AUG-1988
;       UNIX1.PROD              : 23-AUG-1988
;       INSTALL.SH              : 23-AUG-1988
;       !
;
;       UNIX1.PROD
;       - Added support for new workstation
;
;       INSTALL.SH
;       - Modified installation script to check for correct exit
;         status from cpio.
```

The "fix" section is divided into two areas with the ! delimiter separating them. The part before the ! must begin with a "fix" date with form illustrated. The only allowed deviation from the format shown is in the amount of white space found between the **DATE** keyword and the colon and between the colon and the date. The date must have the form DD-MMM-YYYY. One or more product file description lines come after the date line. These must adhere to the same format as the date line. Each file description line can have a different date. The ! delimiter ends the required portion of the "fix" section. The section following is a free-form comment area for describing the features of the new software release.

"Fix" entries are always inserted in the *fixes.com* file immediately following the header section. Thus the most recent software deliveries for a given product are documented near the top of the file.

SEE ALSO

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

NOTES

fixes.com files generally reside on Intergraph delivery systems and are downloaded to Intergraph CLIX systems during product installation for local examination.

NAME

floppypar - partitioned floppy header format

DESCRIPTION

The *mkpar*(1M) command is used to divide a floppy into one or more logical sections called partitions. Information about each partition is contained in the floppy partition header (*floppypar*) that resides on the first block of a partitioned floppy.

The organization of the floppy header is as follows:

Byte Offset	Description
0-3	magic number
4-5	number of logical partitions
6	partition number of logical partition 1
7	modifier number of logical partition 1
8-9	block number of the first block of logical partition 1
10-11	block number of the last block of logical partition 1
12	partition number of logical partition 2
13	modifier number of logical partition 2
14-15	block number of the first block of logical partition 2
16-17	block number of the last block of logical partition 2
18	partition number of logical partition 3
19	modifier number of logical partition 3
20-21	block number of the first block of logical partition 3
22-23	block number of the last block of logical partition 3
...	... ...
(6*n)	partition number of logical partition n
(6*n+1)	modifier number of logical partition n
(6*n+2)-(6*n+3)	block number of the first block of logical partition n
(6*n+4)-(6*n+5)	block number of the last block of logical partition n

where $n < 86$.

The magic number is 0x454e4153.

The number of logical partitions is a 16-bit unsigned integer value.

The partition number and the modifier number of each logical partition are 8-bit unsigned integer values.

The first block and last block values for each logical partition are 16-bit unsigned integer values.

SEE ALSO

mkpar(1M), parck(1M), fl(7S) in the *CLIX System Administrator's Reference Manual.*

NAME

fstab - file system table

DESCRIPTION

The file **/etc/fstab** contains information about file systems for use by *mount*(1M) and *mountall*(1M). Each entry should be specified in the following order:

Order	Entry
1st	block special file name of file system or advertised remote resource
2nd	mount-point directory
3rd	-r if to be mounted read-only; -d[r] if remote
4th	(optional) file system type string
5th	(optional) file system type dependent local file system: backup frequency used by *backup*(1) Network File System (NFS): string of NFS mount options separated by commas Remote File System: ignored

White space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

The following is an example of the file system table:

```
/dev/dsk/s0u0p7.3 /usr  S51K 3
/dev/dsk/s0u0p7.4 /usr/src -r
/dev/dsk/s1u0p7.3 /usr2 5
adv_resource /mnt -d
```

FILES

/etc/fstab

SEE ALSO

mount(1M) in the *CLIX System Administrator's Reference Manual.*
mountall(1M), rmountall(1M) in the *UNIX System V System Administrator's Reference Manual.*

NAME

group - group file

DESCRIPTION

The file */etc/group* contains the following information for each group:

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

This is an ASCII file. The fields are separated by colons; each group is on a separate line. If the password field is null, no password is demanded.

Because of the encrypted passwords, the file can and does have general read permission and can be used, for example, to map numerical group IDs to names.

A group file can have a line beginning with a plus (+) which means to incorporate entries from the Yellow Pages (YP). There are two styles of "+" entries. "+:" means to insert the entire contents of the YP group file at that point. "+*name*" means to insert the entry (if any) for *name* from the YP at that point. If a "+" entry has a non-null password or group member field, the contents of that field will override what is contained in the YP. The numerical group ID field cannot be overridden.

EXAMPLES

```
+myproject:::bill, steve
+:
```

If these entries appear at the end of a group file, the group "myproject" will have members "bill" and "steve", and the password and group ID of the YP entry for the group "myproject". All the groups listed in the YP will be pulled in and placed after the entry for "myproject".

FILES

/etc/group
/etc/yp/group

SEE ALSO

passwd(4).
newgrp(1), passwd(1) in the *UNIX System V User's Reference Manual.*
crypt(3C) in the *UNIX System V Programmer's Reference Manual.*

NAME

hosts - host name database

DESCRIPTION

The *hosts* file contains information regarding the known hosts on the network. For each host a single line should be present with the following information:

official host name
Internet address
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

This file is updated periodically by *namex*(1M), or may be hand edited.

Network addresses are specified in the conventional "." notation using the *inet_addr* routine from the Internet address manipulation library, *inet*(3B). Host names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/hosts

SEE ALSO

gethostbyname(3B), inet(3B),
namex(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

hosts.equiv - host equivalency name database

DESCRIPTION

The *hosts.equiv* file contains a list of hosts. If a user on a remote host has an entry in the local **/etc/passwd** file and the remote host name is in the *hosts.equiv* file, the remote user gains access to the local host. The format of the *hosts.equiv* file is one host name per line.

The *hosts.equiv* file is used by commands such as *rlogin*(1), *rcmd*(1), and *rcp*(1).

The *hosts.equiv* file is not consulted if a user is trying to gain access as root.

FILES

/etc/hosts.equiv

SEE ALSO

rlogin(1).

NAME

JBCFG - optical disk jukebox configuration file

DESCRIPTION

The *JBCFG* configuration file describes the configuration of the Jukebox Interface Management System (JIMS). The file specifies the location of JIMS programs, the program startup options, the terminal device names for the robotics, the generic Small Computer System Interface (SCSI) character device names for the optical drives controlled by the jukeboxes, and logical names for the jukeboxes and the optical drives.

When JIMS is initialized, the configuration file is read to obtain information necessary for the JIMS subtasks to operate successfully. If JIMS cannot understand the configuration file's contents, it will never be successfully initialized. A sample JIMS configuration is included with the optical disk product.

JBCFG can be edited and changed any time, even when JIMS is running. Any changes, however, will not take effect until JIMS is shut down and restarted.

The general format for *JBCFG* is as follows, where *keyword-value* may be a literal or an aggregate consisting of **{**, one or more *keyword* = *value*, and an ending **}**.

Keywords are separated by commas. The major sections of the configuration are as follows:

```
JIMS = {
        keyword-value ...
},
ROUTER = {
        keyword-value ...
},
DATABASE = {
        keyword-value ...
},
JUKEBOX = {
        keyword-value ...
},
OPERATOR = {
        keyword-value ...
},
ERRORLOG = {
        keyword-value ...
}
```

One JUKEBOX entry is required for each jukebox that JIMS manages. Other keywords are nested within this generalized framework, with secondary and tertiary nesting of additional keywords as necessary. Keywords may be in either upper or lowercase, whereas the information following the keyword is case sensitive. A description of the current valid keywords and their legitimate values follows:

JIMS — The valid keyword for the JIMS entry is in the JIMS configuration file as follows:

```
JIMS = {
        checkpoint = ON
},
```

This specifies whether checkpointing is to be turned on or off. Checkpointing consists of writing the time of the request block's current state and the volume database at each change. It is needed with the WARM start capability of JIMS.

ROUTER — The valid keyword for the ROUTER entry is in the JIMS configuration file as follows:

```
ROUTER = {
        filename = RTR.chkpt
},
```

Filename is the path name of the file that contains the request checkpoint file. If no directory is specified, it will default to **/usr/ip32/od**.

DATABASE — The three valid keywords for the DATABASE entry are in the JIMS configuration file as follows:

```
DATABASE = {
        path      = ../bin,
        module    = VRSmain,
        filename  = vdb
},
```

Path describes the directory where the module is located. **Module** is the name of the executable file that maintains the volume resolution subsystem database. **Filename** is the path name of the file that contains the VRS database. If **filename** is only a base name, the database is located in the user's current working directory. **Path** and **module** are required for JIMS to operate. **Filename** defaults to **vdb** if **filename** is not supplied.

JUKEBOX — Four valid keywords for the JUKEBOX entry are in the JIMS configuration file as follows:

```
JUKEBOX = {
    path     = ../bin,
    module   = JBCmain,
    name     = JB01,
    JBC = {
        JBD = {
            path     = ../bin,
            module   = JBDmain
```

```
                },
                JBI = {
                     path      = ../bin,
                     module    = JBImain,
                     device    = /dev/tty02
                },
                filename       = JBC01.ckpt,
                series         = 1800,
                exchange       = NOEXCHANGE,
                mount          = VERIFY,
                unmount        = VERIFY,
                export         = VERIFY,
                drive = {
                     device    = /dev/gs/s3u0,
                     name      = OD001,
                     status    = ONLINE
                },
                drive = {
                     device    = /dev/gs/s3u1,
                     name      = OD002,
                     status    = ONLINE
                }
           }
     },
```

Path and **module** are the same as the **path** and **module** described for database, except that in this context they refer to the Jukebox Control (JBC) module. **Name** is the ASCII name of the jukebox that JIMS uses internally. Currently, JIMS can support 10 jukeboxes.

The JBD entry has two valid keywords: **path** and **module**. The JBI entry has three valid keywords: **path**, **module**, and **device**. The **device** is the name of the tty port that the jukebox robotics is connected to.

The DRIVE entry has three valid keywords: **device**, **name**, and **status**. **Device** is the name of the generic SCSI character device for the optical drives that the jukebox controls. **Name** is the ASCII name of the drive that JIMS uses internally. **Status** is the default status of the drive when JIMS is started. Valid values are ONLINE and OFFLINE.

Other valid keywords in the JUKEBOX JBC entry are as follows:

filename	The name of the checkpoint file used in warm starts.
series	1800.
exchange	NOEXCHANGE is the only valid value keyword at this time.

mount	VERIFY or NOVERIFY is for the optical disk labels when platter mounting is requested.
unmount	VERIFY or NOVERIFY is for the optical disk labels when platter unmounting is requested.
export	VERIFY or NOVERIFY is for the optical disk labels when platter export is requested.

OPERATOR Three valid keywords for the OPERATOR entry are in the JIMS configuration file as follows:

```
OPERATOR = {
      path     = ../bin,
      module   = OPMmain,
      filename = OPMlogfile
},
```

Path and **module** were described previously. **Filename** is the name of the output file containing operator messages.

ERRORLOG Two valid keywords for the ERRORLOG entry are in the JIMS configuration file as follows:

```
ERRORLOG = {
      filename = jimserrorlog,
      debug    = RTR;JBC
}
```

Filename is the name of the output file containing error messages. **Debug** defines the modules that will print debug output. The value of debug can be **all** if all modules are to print debug messages or the first three letters of a module name as follows:

RTR	RTRmain
VRS	VRSmain
OPM	OPMmain
JBC	JBCmain
JBI	JBImain
JBD	JBDmain

If debug messages are needed from more than one module, list the module prefixes separated by semicolons. If **debug** has no value, no debug messages will be written to the error log.

FILES

/dev/gs/*	generic SCSI device files
/usr/ip32/od/JBCFG	juke box configuration file

SEE ALSO

STANDCFG(4).

NAME

kbmap - keyboard map file

DESCRIPTION

The layout of the keyboard can be changed to match the standard keyboard of a particular region by using *kbmap*(1). The file read by *kbmap*(1) contains the information needed to define the eight possible values for each key that can be redefined. The keys that can be redefined are the letter, number, and punctuation keys on the main section of the keyboard. The keypad and function keys cannot be redefined.

The file is arranged in eight groups of 48 hexadecimal numbers, one group for every valid key and qualifier combination. The numbers represent the ASCII codes generated for each key. A -1 is used if a code will not be generated. White space and any line starting with # are ignored.

Each group of hexadecimal numbers must be in the same sequence as the keys on the keyboard. This sequence begins at the top left and contiunes to the lower right by rows. On a North American keyboard, for example, the numbers begin with `, continue right to =, move down to Q on the next row, and eventually end with /. At the end of this sequence, 48 keys have been specified.

The code specifications must be grouped in the following order:

unshifted keys
unshifted control keys
unshifted alternate keys
unshifted caps lock keys
shifted keys
shifted control keys
shifted alternate keys
shifted caps lock keys

The groups do not need to be separated by delimiters.

EXAMPLES

```
# Denmark keymap

# Unshifted
0x7e 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x30 0x2b 0x27
0x71 0x77 0x65 0x72 0x74 0x79 0x75 0x69 0x6f 0x70 0xe5 0x22
0x61 0x73 0x64 0x66 0x67 0x68 0x6a 0x6b 0x6c 0xe6 0xf8 0x27
0x3c 0x7a 0x78 0x63 0x76 0x62 0x6e 0x6d 0x2c 0x2e 0x2d

# Unshifted, Ctrl
0x7e 0x31 0x00 0x33 0x34 0x35 0x1e 0x37 0x38 0x39 0x30 0x2b 0x27
0x11 0x17 0x05 0x12 0x14 0x19 0x15 0x09 0x0f 0x10 0x1d 0x22
0x01 0x13 0x04 0x06 0x07 0x08 0x0a 0x0b 0x0c 0x00 0x23 0x27
0x3c 0x1a 0x18 0x03 0x16 0x02 0x0e 0x0d 0x2c 0x2e 0x1f
```

```
# Unshifted, Alt Mode
0xe1 0xe2 0xe4 0xe6 0xe8 0xea 0xec 0xee 0xf0 0xf2 0xf4 0x1c 0xc0
0xc7 0xc9 0xcb 0xcc 0xce 0xd0 0xd2 0xd4 0xd6 0xd8 0x5d 0xc1
0x9d 0x9f 0xa1 0xa3 0xa5 0xa7 0xa8 0xba 0xbb 0x00 0x23 0xc0
0x89 0x8b 0x8c 0x8e 0x90 0x92 0x93 0x95 0x97 0x99 0xfb
# Unshifted, Caps Lock
0x7e 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x30 0x2b 0x27
0x51 0x57 0x45 0x52 0x54 0x59 0x55 0x49 0x4f 0x50 0xc5 0x22
0x41 0x53 0x44 0x46 0x47 0x48 0x4a 0x4b 0x4c 0xc6 0xd8 0x27
0x3c 0x5a 0x58 0x43 0x56 0x42 0x4e 0x4d 0x2c 0x2e 0x2d
# Shifted
0xb0 0x21 0x22 0xa7 0x24 0x25 0x26 0x2f 0x28 0x29 0x3d 0x3f 0x60
0x51 0x57 0x45 0x52 0x54 0x59 0x55 0x49 0x4f 0x50 0xc5 0x5e
0x41 0x53 0x44 0x46 0x47 0x48 0x4a 0x4b 0x4c 0xc6 0xd8 0x2a
0x3e 0x5a 0x58 0x43 0x56 0x42 0x4e 0x4d 0x3b 0x3a 0x5f
# Shifted, Ctrl
0x00 0x21 0x22 0x00 0x24 0x25 0x26 0x2f 0x28 0x29 0x3d 0x3f 0x60
0x11 0x17 0x05 0x12 0x14 0x19 0x15 0x09 0x0f 0x10 0x1b 0x1e
0x01 0x13 0x04 0x06 0x07 0x08 0x0a 0x0b 0x0c 0x00 0x1c 0x2a
0x3e 0x1a 0xa8 0x03 0x16 0x02 0x0e 0x0d 0x3b 0x3a 0x1f
# Shifted, Alt Mode
0x00 0xe3 0xc1 0x40 0xe9 0xeb 0xef 0x9b 0xf3 0xfa 0x06 0x9c 0xe0
0xc8 0xca 0xcb 0xcd 0xcf 0xd1 0xd3 0xd5 0xd7 0xd8 0x5b 0xed
0x9e 0xa0 0xa2 0xa4 0xa6 0xa7 0xa9 0xba 0xbc 0x00 0x5c 0xf1
0x8a 0x8b 0x8c 0x8f 0x91 0x92 0x94 0x96 0xbe 0xbf 0x02
# Shifted, Caps Lock
0xb0 0x21 0x22 0x40 0x24 0x25 0x26 0x2f 0x28 0x29 0x3d 0x3f 0x60
0x51 0x57 0x45 0x52 0x54 0x59 0x55 0x49 0x4f 0x50 0xc5 0x5e
0x41 0x53 0x44 0x46 0x47 0x48 0x4a 0x4b 0x4c 0xc6 0x5c 0x2a
0x3e 0x5a 0x58 0x43 0x56 0x42 0x4e 0x4d 0x3b 0x3a 0x5f
```

FILES

/usr/lib/kbmap/*	files for each keyboard type

SEE ALSO

kbmap(1).

NAME

limits - file header for implementation-specific constants

SYNOPSIS

#include <limits.h>

DESCRIPTION

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

```
#define ARG_MAX      5120                     /* max length of arguments to exec */
#define CHAR_BIT     8                        /* # of bits in a "char" */
#define CHAR_MAX     255                      /* max integer value of a "char" */
#define CHAR_MIN     0                        /* min integer value of a "char" */
#define CHILD_MAX    25                       /* max # of processes per user ID */
#define CLK_TCK      100                      /* # of clock ticks per second */
#define DBL_DIG      15                       /* digits of precision of a "double" */
#define DBL_MAX      1.79769313486231470e+308 /* max decimal value of a "double" */
#define DBL_MIN      4.94065645841246544e-324 /* min decimal value of a "double" */
#define FCHR_MAX     1048576                  /* max size of a file in bytes */
#define FLT_DIG      7                        /* digits of precision of a "float" */
#define FLT_MAX      3.40282346638528860e+38  /* max decimal value of a "float" */
#define FLT_MIN      1.40129846432481707e-45  /* min decimal value of a "float" */
#define HUGE_VAL     3.40282346638528860e+38  /* error value returned by Math lib */
#define INT_MAX      2147483647               /* max decimal value of an "int" */
#define INT_MIN      -2147483648              /* min decimal value of an "int" */
#define LINK_MAX     1000                     /* max # of links to a single file */
#define LONG_MAX     2147483647               /* max decimal value of a "long" */
#define LONG_MIN     -2147483648              /* min decimal value of a "long" */
#define NAME_MAX     14                       /* max # of characters in a file name */
#define OPEN_MAX     20                       /* max # of files a process can have open */
#define PASS_MAX     8                        /* max # of characters in a password */
#define PATH_MAX     256                      /* max # of characters in a path name */
#define PID_MAX      30000                    /* max value for a process ID */
#define PIPE_BUF     5120                     /* max # bytes atomic in write to a pipe */
#define PIPE_MAX     5120                     /* max # bytes written to pipe in a write */
#define SHRT_MAX     32767                    /* max decimal value of a "short" */
#define SHRT_MIN     -32768                   /* min decimal value of a "short" */
#define STD_BLK      1024                     /* # bytes in a physical I/O block */
#define SYS_NMLN     9                        /* # of chars in uname-returned strings */
#define UID_MAX      60000                    /* max value for a user or group ID */
#define USI_MAX      4294967295               /* max decimal value of an "unsigned" */
#define WORD_BIT     32                       /* # of bits in a "word" or "int" */
```

NAME

master - master configuration database

DESCRIPTION

The *master* configuration database is a collection of files that contain configuration information for devices and modules included in the system. This collection of files is contained in several directories. The default directories are **/usr/src/uts/clipper/master.d/***machine*, where *machine* is the name of the machine for which the files apply. A file is named with the module name to which it applies.

The files are used by two programs. Consequently, there are two sets of directives contained in the files. One program is *mkconfig*(1M), which uses the files to generate a configuration file. The other is *sysconfig*(1M), which obtains configuration data used to allow the configuration and values in the files to be changed.

The set of directives understood by *mkconfig*(1M) are as follows:

$[[-]*name1* [**&&** [-]*name2*]]**:**
$[[-]*name1* [|| [-]*name2*]]**:**

Name a section of a file. The named section begins on the line after the directive and ends either at the end of the current file or before another named section or macro begins. The null section name **$:** may also be used to end a named section. Named sections may not be nested. A named section with its name prepended with the minus sign (-) is output only if file *name* is not in the argument list. A named section with a logical OR (||) or AND (&&) of another named section is output only if the logical condition results in a TRUE value. A section with the special name **_end** is output only after the last of any generated configuration tables.

$name([*arg1*, *arg2*, ... *argn*])**:**

Define a simple macro with optional arguments. A macro definition is ended as a named section is. A macro is much like a named section except that it is output only if it is invoked by a **MACRO** directive or appears in the **model** specifier of the **VECTOR** directive. Any occurrence of the argument strings in the body of the macro are replaced by the argument strings of the invoking instance of the macro. Macro bodies may not contain macro references or new macro definitions.

$MACRO *macroname*(*arg1*, *arg2*, ... *argn*)

Invoke a defined macro. The argument count must match the argument count of the defined macro. The body of the macro is output with invoking arguments substituted for defined arguments.

$*name section*

Write *section* to the output file if the file *name* is present in the argument list and it contains a *section* segment.

$* *section*
  If any files in the argument list contain a named *section*, these files are copied to the output file.

$DEVICE [b|c]major=*n*|*n-m* **prefix**=*string*
$DEVICE [b|c]major=*n*|*n-m* *identifier*=*name* ...
  Add an entry to a device table. **Cmajor** specifies the *cdevsw* table and **bmajor** specifies the *bdevsw* table. The major number can be a single number or a span of major numbers, such as **0-7**. **Prefix** takes a string as an argument and is used to form the standard names for the switch table entries. For nonstandard names, the device directive can explicitly specify each entry in the *cdevsw* or *bdevsw* table. After the major number is specified, the remainder of the line may have *identifiers* followed by = and the nonstandard name. The valid **cmajor** *identifiers* are **open**, **close**, **read**, **write**, **ioctl**, **tty**, and **stream**. The valid **bmajor** *identifiers* are **open**, **close**, **strategy**, and **print**. More than one **DEVICE** line may exist for the same major slot as long as specific table entries are not redefined.

$TTY_DEVICE cmajor=*n*|*n-m* **prefix**=*string*
$TTY_DEVICE cmajor=*n*|*n-m* *identifier*=*name* ...
  Identical to the **DEVICE** directive, but allows automatic generation of the tty field for terminal devices.

$XIO_DEVICE xmajor=*n*|*n-m* **prefix**=*string*
$XIO_DEVICE xmajor=*n*|*n-m* *identifier*=*name* ...
  Identical to the **DEVICE** directive, but adds the entry to the *xdevsw* table, and the valid **xmajor** *identifiers* are **init**, **start**, **finish**, **exit**, **exec**, and **fork**.

$STREAM_DEVICE cmajor=*n*|*n-m* **prefix**=*string*
  Identical to the **DEVICE** directive, but allows automatic generation of the stream field for stream devices.

$STREAM_MODULE name=*string1* **module**=*string2*
  Identical to the **$DEVICE** directive, but adds entries to the *fmodsw*[] array.

$FILESYSTEM prefix=*s1* **flags**=*s2* **pipe**=*s3* **name**=*s4* **notify**=*s5*
  Add an entry to the file system switch table *fstypsw*[] and a corresponding entry to the *fsinfo*[] table. The arguments are taken as strings and are inserted in the tables as is. The **prefix** directive is used to form the prefixes to the standard entry points in the *fstypsw*[] table.

$VECTOR model=*macro*(*arg1*, ..., *argn*) **service**=*name* **vector**=*g,l* **ssw**=*val*
  Specify the interrupt service linkages for a given vector. **model** names a macro and its optional arguments to be used in generating the assembly language stub that calls the kernel interrupt service routine. **Service** specifies the name of the address that will be written into the actual interrupt vector. The service routine name is typically generated by the macro specified by the **model** directive.

Vector may be specified as *group,level* or as a single address specifying the physical address of the interrupt vector. Vector addresses are understood to be hexadecimal when preceeded with a leading '0x'. **Ssw** is the value that specifies the new value of the System Status Word (SSW) for the specified vector. *Val* must be a decimal or hexadecimal constant or the name of a constant defined earlier as by the C preprocessor.

$FAMILY code=*family-code* **bname=***board-name* **probe=***routine*
Specify the startup routine for a given Shared Resource (SR) Bus board. **Code** provides the family code number of the board and must be given in hexadecimal format. **Bname** specifies the name of the board. The name may be more than one word and is delimited by the **probe** keyword. **Probe** is the routine that will be called to initialize the board.

$LOAD name=*s1* [**init=***s2*] **objects=***s3*[(*s4*[,*s5*])] ...
Identify the code that will be loaded in memory upon demand. **Name** takes a string as an argument and is used to name the section of memory that will be reserved. **Init** takes as an argument the name of the initialization routine that will be executed immediately after the code is loaded into memory. **Objects** specifies the object files that will be loaded. An entire library may be specified by the name of the library. If only certain objects in a library are needed, they are specified by a comma-separated list enclosed in parentheses following the library name. The list may not contain white space. There must be a separate **objects** identifier for each library referenced. **Objects** identifiers may be separated by either spaces or tabs.

The set of directives understood by *sysconfig*(1M) are as follows:

***$CONFIG** [*category* **$IN** | **$OUT** | **$NOLIST**] [**$ADVANCED**]
***$CONFIG** [**$NOLIST**] [**$ADVANCED**]
Identifies the file as a configuration file. It must be the first line in the file. If it is not found, the file is ignored.

If *category* is specified, all parameters in the file are grouped together and given a name that is the same as the file name. This group is then placed in the menu category *category*. If no *category* is specified, the categories in the ***PAR** directives are used. If *category* contains any spaces, it must be enclosed within single or double quotation marks.

$IN specifies that the default is to include the file when a system is made.

$OUT specifies that the default is not to include the file when a system is made.

$NOLIST specifies that the file is never to be included when a system is made.

If the line ***$CONFIG** or the line ***$CONFIG $ADVANCED** is used, the file will always be included.

$ADVANCED specifies that the file and any parameters in it have an advanced nature.

Up to three lines of descriptive help, each 80 characters long, may be included. The help lines must immediately follow the ***CONFIG** directive and have an asterisk (*) as the first character. To mark the end of the ***$CONFIG** help lines, the line

***$$**

must immediately follow them.

***$PAR** [*category*] *default* [**$ADVANCED**]

The ***PAR** directive identifies a "#define" parameter as being tunable. It must be immediately followed by up to three descriptive help lines, and then by the "#define" statement.

Category specifies the category that the parameter should be placed under. This category will be the default for all following ***PAR** directives until another *category* is specified. If a *category* was specified in the ***CONFIG** directive, the ***PAR** *category* option is ignored. If neither the ***CONFIG** directive nor any ***PAR** directives specify a *category*, *category* defaults to the name of the file. If *category* contains any spaces, it must be enclosed within single or double quotation marks.

Default is the default value of the parameter. If it contains any spaces, it must be enclosed within single or double quotatation marks.

$ADVANCED specifies that the parameter has an advanced nature. If the **ADVANCED** option is included in the ***CONFIG** directive, all parameters in the file automatically default to advanced.

EXAMPLES

This is a sample file showing the use of some of the directives for both *mkconfig*(1M) and *sysconfig*(1M).

```
*$CONFIG "Inter-process Communication" $IN $ADVANCED
*The XYZ terminal driver. The /dev entries associated
*with this driver are /dev/ttxyz??.
*$$

$includes:
#include "sys/xio/xyz.h"

$defines:
*$PAR 20
*The total number of xyz terminals available on the system.
#define NXYZ        20

$code:
int     nxyz = NXYZ;
```

```
struct   tty xyz_tty[NXYZ];
struct   xyzproc xyzproc[NXYZ];

$TTY_DEVICE cmajor=25 prefix=xyz

$XIO_DEVICE xmajor=2 start=xyz_start finish=xyz_finish

$FAMILY code=0x2a  bname=XYZ Communication Board  probe=xyz_probe

$LOAD name=xyz init=xyz_init objects=../lib.io(xyz.o)
```

FILES

/usr/src/uts/clipper/master.d/*machine*/*   configurable kernel files

SEE ALSO

mkconfig(1M), sysconfig(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

mnttab - mounted file system table

SYNOPSIS

#include <mnttab.h>

DESCRIPTION

The file **/etc/mnttab** contains an entry for each device mounted by *mount*(1M). Each entry is a structure in **<mnttab.h>** defined as follows:

```
struct mnttab {
        char      mt_dev[32];
        char      mt_filsys[32];
        short     mt_ro_flg;
        time_t    mt_time;
        char      mt_fstyp[16];
        char      mt_mntopts[64];
};
```

Each entry is 150 bytes in length. The members of the structure are as follows:

mt_dev	The null-padded name of the place where the special file is mounted.
mt_filsys	The null-padded root name of the mounted special file.
mt_ro_flg	The mounted special file's read/write permissions.
mt_time	The date on which the special file was mounted.
mt_fstyp	The null-padded name of file system type.
mt_mntopts	The null-padded string of mount options. The mount options are only used in the case of a Network File System.

SEE ALSO

mount(1M) in the *CLIX System Administrator's Reference Manual.*
setmnt(1M) in the *UNIX System V System Administrator's Reference Manual.*

NAME

networks - network name database

DESCRIPTION

The file */etc/networks* contains information regarding the known networks which comprise the Defense Advanced Research Project Agency (DARPA) Internet. For each network a single line should be present with the following information:

official network name
network number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. It will be necessary to make local changes to this file to bring it up to date regarding unofficial aliases and/or unknown networks for a specific site.

Network number may be specified in the conventional "." notation using the *inet_network* routine from the Internet address manipulation library, *inet*(3B). Network names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/networks

SEE ALSO

getnetent(3B), inet(3B).

NAME

passwd - password file

DESCRIPTION

The file */etc/passwd* contains for each user the following information:

name	User's login name—contains no uppercase characters and must not be greater than eight characters long.
password	Encrypted password.
numerical user ID	This is the user's ID in the system and it must be unique.
numerical group ID	This is the number of the group that the user belongs to.
user's real name	In some versions of UNIX, this field also contains the user's office, extension, home phone, and so on. For historical reasons this field is called the GCOS field.
initial working directory	The directory that the user is positioned in when they log in—this is known as the home directory.
shell	Program to use as shell when the user logs in.

The user's real name field may contain "&", meaning insert the login name. The password file is an ASCII file. Each field within each user's entry is separated from the next by a colon. Each entry is on a separate line. If the password field is null, no password is demanded; if the shell field is null, **/bin/sh** is used.

The *passwd* file can also have line beginning with a plus (+), which means to incorporate entries from the Yellow Pages (YP). There are three styles of "+" entries. "+" means to insert the entire contents of the YP password file at that point; "+*name*" means to insert the entry (if any) for *name* from the YP at that point; "+@*name*" means to insert the entries for all members of the network group *name* at that point. If a "+" entry has a non-null password, directory, GCOS, or shell field, it overrides what is contained in the YP. The numerical user ID and group ID fields cannot be overridden.

EXAMPLES

Here is a sample **/etc/passwd** file:

```
root:q.mJzTnu8icF.:0:10:Administrator:/:/bin/ksh
tut:6k/7KCFRPNVXg:508:10:Bill Tuthill:/usr2/tut:/bin/ksh
+john:
+@documentation:no-login:
+:::Guest
```

In this example, there are specific entries for users "root" and "tut", in case the YP are out of order. The user "john" will have his password entry in the YP incorporated without change; anyone in the netgroup *documentation* will have their password field disabled, and anyone else will be able to log in with their usual password, shell, and home directory, but with a GCOS field of "Guest".

Because of the encrypted passwords, the file has general read permission and can be used, for example, to map numerical user IDs to names.

Appropriate precautions must be taken to lock the **/etc/passwd** file against simultaneous changes if it is to be edited with a text editor.

FILES

/etc/passwd

SEE ALSO

getpwent(3C), group(4).
login(1), passwd(1) in the *UNIX System V User's Reference Manual.*
crypt(3C) in the *UNIX System V Programmer's Reference Manual.*

NAME

printcap - BSD printer capability database

DESCRIPTION

printcap is a printer capability database used to describe line printers. The spooling system accesses the *printcap* file every time it is used, allowing dynamic addition and deletion of printers. Each entry in the database describes one printer.

The environment variable PRINTER may be used to specify a printer. Each spooling utility supports an option, **-P** *printer*, to allow explicit naming of a destination printer.

Refer to the "BSD LP Spooler Tutorial" in the *CLIX System Guide* for a complete discussion on how to set up the database for a given printer.

Capabilities

Name	Type	Default	Description
af	str	NULL	name of accounting file
br	num	none	if lp is a tty, set the baud rate (*ioctl*(2) call)
cf	str	NULL	cifplot data filter
df	str	NULL	*TeX*™ data filter (DVI format)
ff	str	\f	string to send for a form feed
fo	bool	false	print a form feed when device is opened
gf	str	NULL	graph data filter (*plot*(3X) format)
hl	bool	false	print the burst header page last
ic	bool	false	supports (nonstandard) *ioctl*(2) to indent printout
if	str	NULL	name of text filter which does accounting
lf	str	/dev/console	error logging file name
lo	str	lock	name of lock file
lp	str	/dev/lp	device name to open for output
mx	num	1000	max file size (in BUFSIZ blocks), zero=unlimited
nd	str	NULL	next directory for list of queues (unimplemented)
nf	str	NULL	*ditroff* data filter (device-independent troff)
of	str	NULL	name of output filtering program
pc	num	200	price per foot or page in hundredths of cents
pl	num	66	page length (in lines)
pw	num	132	page width (in characters)
px	num	0	page width in pixels (horizontal)
py	num	0	page length in pixels (vertical)
rf	str	NULL	filter for printing FORTRAN-style text files
rg	str	NULL	restricted group, only members allowed access
rm	str	NULL	machine name for remote printer
rp	str	lp	remote printer name argument
rs	bool	false	restrict remote users to those with local accounts
rw	bool	false	open the printer device for reading and writing
sb	bool	false	short banner (one line only)
sc	bool	false	suppress multiple copies
sd	str	/usr/spool/lpd	spool directory
sf	bool	false	suppress form feeds

sh	bool	false	suppress printing of burst page header
st	str	status	status file name
tf	str	NULL	*troff* data filter (cat phototypesetter)
tr	str	NULL	trailer string to print when queue empties
vf	str	NULL	raster image filter
Cc	num	0	clear control modes (*termio*(7S))
Cs	num	0	set control modes (*termio*(7S))
Ic	num	0	clear input modes (*termio*(7S))
Is	num	0	set input modes (*termio*(7S))
Lc	num	0	clear local modes (*termio*(7S))
Ls	num	0	set local modes (*termio*(7S))
Oc	num	0	clear output modes (*termio*(7S))
Os	num	0	set output modes (*termio*(7S))

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

The following is a sample entry from a *printcap* file.

```
lp | local line printer:
        :lp=/dev/lp:sd=/usr/spool/lpd:lf=/usr/adm/lpd-errs:
```

Entries may continue on to multiple lines by giving a "\" as the last character of a line, and empty fields may be included for readability (here between the last field on a line and the first field on the next). Comments may be included on lines beginning with "#".

Filters

The *lpd*(1M) daemon creates a pipeline of *filters* to process files for various printer types. The filters selected depend on the flags passed to *lpr*(1). The pipeline setup is as follows:

-p	pr | if	regular text + *pr*(1)
none	if	regular text
-c	cf	cifplot
-d	df	DVI (*TeX*)
-g	gf	*plot*(3)
-n	nf	*ditroff*
-f	rf	FORTRAN
-t	tf	*troff*
-v	vf	raster image

The **if** filter is invoked with the following arguments:

if [**-c**] **-w***width* **-l***length* **-i***indent* **-n** *login* **-h** *host acct-file*

-c is passed only if the **-l** flag (pass control characters literally) is specified to *lpr*. *Width* and *length* specify the page width and length (from **pw** and **pl** respectively) in characters. The **-n** and **-h** parameters specify the login name and host name of the owner of the job, respectively. *Acct-file* is passed from the **af** *printcap* entry.

If no **if** is specified, **of** is used instead. However, **of** is opened only once, while **if** is opened for every job. Thus, **if** is better suited for performing accounting. The **of** is only given the *width* and *length* flags.

All other filters are called as follows:

filter **-x***width* **-y***length* **-n** *login* **-h** *host acct-file*

Width and *length* are represented in pixels, specified by the **px** and **py** entries, respectively.

All filters take **stdin** as the file, **stdout** as the printer, will log to **stderr**, and must not ignore SIGINT.

SEE ALSO

lpr(1), lpq(1), lprm(1).
lpc(1M), lpd(1M), pac(1M) in the *CLIX System Administrator's Reference Manual.*
"BSD LP Spooler Tutorial" in the *CLIX System Guide.*

NAME

protocols - protocol name database

DESCRIPTION

The file */etc/protocols* contains information regarding the known protocols used in the Defense Advanced Research Project Agency (DARPA) Internet. For each protocol, a single line should be present with the following information:

official protocol name
protocol number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file.

Protocol names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/protocols

SEE ALSO

getprotoent(3B).

NAME

rcsfile - format of RCS file

DESCRIPTION

A Revision Control System (RCS) file is an ASCII file. Its contents are described by the grammar below. The text is free format. (Spaces, tabs, and newlines have no significance except in strings.) Strings are enclosed by "@". If a string contains a "@", it must be doubled.

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

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

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

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

All <delta> nodes whose numbers consist of 2n fields ($n \geqslant 2$) (such as 3.1.1.1, 2.1.2.2, etc.) are linked as follows. All nodes whose first (2n)-1 number fields are identical are linked through the **next** field in the order of increasing numbers. For each such sequence, the <delta> node whose number is identical to the first 2(n-1) number fields of the deltas on that sequence is called the branchpoint. The **branches** field of a node contains a list of the numbers of the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers.

Example:

```
                           Head
                             |
                             |
                             v
                         ---------
   / \          / \      |       |      / \           / \
  /   \        /   \     |  2.1  |     /   \         /   \
 /     \      /     \    |       |    /     \      /       \
/1.2.1.3\    /1.3.1.1\   |       |   /1.2.2.2\   /1.2.2.1.1.1\
---------    ---------   ---------   ---------   -------------
    ^            ^           |           ^             ^
    |            |           |           |             |
    |            |           v           |             |
   / \           |       ---------      / \            |
  /   \          |       \  1.3  /     /   \           |
 /     \         ---------\     /     /     \-----------
/1.2.1.1\                  \   /     /1.2.2.1\
---------                   \ /      ---------
    ^                        |           ^
    |                        |           |
    |                        v           |
    |                    ---------       |
    |                    \  1.2  /       |
    ----------------------\     /---------
                           \   /
                            \ /
                             |
                             |
                             v
                         ---------
                         \  1.1  /
                          \     /
                           \   /
                            \ /
```

Fig. 1: A revision tree

SEE ALSO

ci(1), co(1), ident(1), rcs(1), rcsclean(1), rcsdiff(1), rcsmerge(1), rlog(1).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision

Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

IDENTIFICATION

Author: Walter F. Tichy,
Purdue University, West Lafayette, IN 47907.
Copyright © 1982 by Walter F. Tichy.

NAME

reloc - relocation information for a common object file

SYNOPSIS

#include <reloc.h>

DESCRIPTION

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

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

#define R_ABS       0
#define R_RELBYTE   017
#define R_RELWORD   020
#define R_RELLONG   021
#define R_PCRBYTE   022
#define R_PCRWORD   023
#define R_PCRLONG   024
```

As the link editor reads each input section and performs relocation, the relocation entries are read. They direct the treatment of references found within the input section.

R_ABS	The reference is absolute and no relocation is necessary. The entry will be ignored.
R_RELBYTE	A direct 8-bit reference to the symbol's virtual address.
R_RELWORD	A direct 16-bit reference to the symbol's virtual address.
R_RELLONG	A direct 32-bit reference to the symbol's virtual address.
R_PCRBYTE	A "PC-relative" 8-bit reference to the symbol's virtual address. The actual address is calculated by adding a constant to the PC value.
R_PCRWORD	A "PC-relative" 16-bit reference to the symbol's virtual address. The actual address is calculated by adding a constant to the PC value.
R_PCRLONG	A "PC-relative" 32-bit reference to the symbol's virtual address. The actual address is calculated by adding a constant to the PC value.

A relocation entry with 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 relocation address.

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

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

SEE ALSO

as(1), ld(1), a.out(4).
syms(4) in the *UNIX System V Programmer's Reference Manual.*

NAME

.rhosts - remote user access list

DESCRIPTION

A *.rhosts* file in a user's home directory allows users on remote hosts to gain access as the local user. The format for the file is as follows:

host-name [*user-name*]

Items are separated by any number of blanks and/or tab characters. If a user name is not specified, all users on the specified host may gain access. If a user name is specified then only that user on the given host may gain access.

Either the super-user or the local user must own the *.rhosts* file.

FILES

~/.rhosts

SEE ALSO

rlogin(1).

NAME

rmtab - remotely mounted NFS file system table

DESCRIPTION

The file */etc/rmtab* contains a record of all clients who have performed remote mounts of Network File Systems (NFS) from the server machine. Whenever a remote *mount*(1M) is done, an entry is made in the *rmtab* file of the machine serving that file system. The table is a series of lines with the following form:

host-name:directory

This table is used only to preserve information between crashes, and is read only by *mountd*(1M) when it starts up. *mountd*(1M) keeps an in-core table, which it uses to handle requests from programs like *showmount*(1) and *shutdown*(1M).

FILES

/etc/rmtab

SEE ALSO

showmount(1M), mountd(1M), mount(1M) in the *CLIX System Administrator's Reference Manual.*
shutdown(1M) in the *UNIX System V System Administrator's Reference Manual.*

BUGS

Although the *rmtab* table is close to the truth, it is not always 100% accurate. It is removed each time the server is rebooted to clean up lingering mount entries.

When a remotely mounted file system is unmounted using *umount*(1M), the entry is removed.

NAME

rpc - RPC program number database

DESCRIPTION

The Remote Procedure Call (RPC) file contains user readable names that can be used in place of RPC program numbers. Each line has the following information:

name of server for the RPC program
RPC program number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

FILES

/etc/rpc

SEE ALSO

getrpcent(3R).

NAME

server.dat - XNS server information file

DESCRIPTION

server.dat contains information about Xerox Network System (XNS) servers on the local machine. *server.dat* allows client and server programs to be linked together. The *sernum* and *server* arguments to *sni_connect*(3N) are used to access the proper server entry in *server.dat*.

Every server started by the *xns_listener*(1M) has one entry in *server.dat*. Each entry in *server.dat* has four fields delimited by **!**, with the first and the last character of the entry also containing a **!**. Each entry contains the following information:

server number
flags
server path and arguments
default login name

The server number is a value from 0-32767, inclusive, which corresponds to *sernum*. The flags allow the *xns_listener*(1M) to control access to a server. The following flags are supported:

U	require user name
P	require password
D	use default user name
N	disallow null passwords

If a user name or a password is required, the *xns_listener*(1M) will not start the server unless *sni_connect*(3N) gives a proper user name and/or password. If the flags specify a default user name, the default login name is used. If *sni_connect*(3N) specifies a *sernum* of 0, the *server* argument is used as the path to the server. Otherwise, the server path in the *server.dat* file is used.

EXAMPLES

An example *server.dat* file is as follows:

```
!0!UP!!!
!6!UP!/usr/ip32/inc/fmus!!
!7!D!/usr/ip32/inc/rtape_s!root!
!1000!D!/usr/joe/sni/server!joe!
```

FILES

/usr/ip32/inc/server.dat server list

SEE ALSO

sni_connect(3N), sni_accept(3N).
xns_listener(1M) in the *CLIX System Administrator's Reference Manual*.
"XNS Network Programming Tutorial" in the *CLIX System Guide*.

NOTES

Server number 0 can be useful for debugging.

The *xns_listener*(1M) reads *server.dat* every time a server is started. It does not need to be restarted to know about new server entries.

WARNINGS

Intergraph reserves server numbers 0-999.

NAME

services - service name database

DESCRIPTION

The file */etc/services* contains information regarding the known services available in the Defense Advanced Research Project Agency (DARPA) Internet. For each service, a single line should have the following information:

official service name
port number
protocol name
aliases

Items are separated by any number of blanks and/or tab characters. The port number and protocol name are considered a single item; a "/" is used to separate the port and protocol (such as 512/tcp). A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file.

Service names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/services

SEE ALSO

getservent(3B).

NAME

STANDCFG - optical disk standalone configuration file

DESCRIPTION

STANDCFG describes the configuration of the optical disk file system. The file specifies the generic Small Computer System Interface (SCSI) character device names for the optical drives not controlled by an optical disk jukebox. Each entry must appear on a separate line.

EXAMPLES

```
/dev/gs/s3u0
/dev/gs/s4u0
```

FILES

/dev/gs/*

SEE ALSO

JBCFG(4).

NAME

statmon: record, recover, state - status daemon directory and file formats

SYNOPSIS

/etc/sm/record

/etc/sm/recover

/etc/sm/state

DESCRIPTION

record and *recover* are text files generated by *statd*(1M). Each host name in *record* represents the name of a machine to be monitored by *statd*(1M). Each host name in *recover* represents the name of a machine to be notified by *statd*(1M) upon its recovery from interruption of locking services.

state is a text file generated by *statd*(1M) to record the status daemon's current version number. This version number is incremented each time a crash or recovery occurs.

SEE ALSO

statd(1M), lockd(1M) in the *CLIX System Administrator's Reference Manual.*

NAME

ypfiles - the YP database and directory structure

DESCRIPTION

The Yellow Pages (YP) network lookup service uses a database of *ndbm* files in the directory hierarchy at **/etc/yp**. An *ndbm* database consists of two files, created by calls to the *dbm*(3B) library package. One has the file name extension ".pag" and the other has the file name extension ".dir". For instance, the database named **hosts.byname**, is implemented by the pair of files **hosts.byname.pag** and **hosts.byname.dir**. An *ndbm* database served by the YP is called a YP *map*. A YP *domain* is a named set of YP maps. Each YP domain is implemented as a subdirectory of **/etc/yp** containing the map. Any number of YP domains can exist. Each may contain any number of maps.

No maps are required by the YP lookup service itself, although they may be required for the normal operation of other parts of the system. There is not a list of maps which YP serves—if the map exists in a given domain, and a client asks about it, the YP will serve it. For a map to be accessible consistently, it must exist on all YP servers that serve the domain. To provide data consistency between the replicated maps, an entry to run *ypxfr*(1M) periodically should be made in *cron*(1M) on each server. More information on this topic is in *ypxfr*(1M).

YP maps should contain two distinguished key-value pairs. The first is the key YP_LAST_MODIFIED, having as a value a ten-character ASCII order number. The order number should be the CLIX time (in seconds) when the map was built. The second key is YP_MASTER_NAME, with the name of the YP master server as a value. *makedbm*(1M) generates both key-value pairs automatically. A map that does not contain both key-value pairs can be served by the YP, but the *ypserv*(1M) process will not be able to return values for "Get order number" or "Get master name" requests. In addition, values of these two keys are used by *ypxfr*(1M) when it transfers a map from a master YP server to a slave. If *ypxfr*(1M) cannot figure out where to get the map, or if it is unable to determine whether the local copy is more recent than the copy at the master, extra command line switches must be set when it is run.

YP maps must be generated and modified only at the master server. They are copied to the slaves using *ypxfr*(1M) to avoid potential byte-ordering problems among YP servers running on machines with different architectures, and to minimize the amount of disk space required for the *ndbm* files. The YP database can be initially set up for both masters and slaves by using *ypinit*(1M).

After the server databases are set up, it is probable that the contents of some maps will change. In general, some ASCII source version of the database exists on the master, and it is changed with a standard text editor. The update is incorporated into the YP map and is propagated from the master to the slaves by running **/etc/yp/Makefile**. All supplied maps have entries in

/etc/yp/Makefile. If a YP map is added, this file should be edited to support the new map. The makefile uses *makedbm*(1M) to generate the YP map on the master, and *yppush*(1M) to propagate the changed map to the slaves. *yppush*(1M) is a client of the map *ypservers*(1M), which lists all the YP servers (see *yppush*(1M)).

SEE ALSO

makedbm(1M), ypinit(1M), ypmake(1M), ypxfr(1M), yppush(1M), yppoll(1M), ypserv(1M), rpcinfo(1M) in the *CLIX System Administrator's Reference Manual*.

NAME

ypmapxlate - translation table to handle long map names

DESCRIPTION

A map name *X* under domain *Y* exists as the two files named *X*.**pag** and *X*.**dir**, both under the directory **/etc/yp/***Y*. Thus, the length of the name *X* can be no more than ten characters, since ten added to the length of the extension ".pag" or ".dir" is equal to 14, which is the maximum file name length. A critical map name such as **passwd.byname** will clearly be a problem to represent.

The file **/etc/yp/YP_MAP_X_LATE** contains entries of the following form:

```
long_map_name        short_map_name
```

When the Yellow Pages (YP) server receives a packet containing a logical (long) map name, this file is looked up to determine the physical (short) map name. Conversely, when the server transmits a packet that will contain a map name, the physical map name is looked up in the translate file to find the logical name for transmission.

FILES

/etc/yp/YP_MAP_X_LATE

SEE ALSO

ypfiles(4).

BUGS

Comments (#) cannot be put in the map file.

Miscellaneous (5)

NAME

intro - introduction to miscellaneous facilities

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, and include file definitions.

NAME

fcntl - file control options

SYNOPSIS

```
#include <fcntl.h>
#include <sys/file.h>
```

DESCRIPTION

The *fcntl*(2) function provides for control over open files. These include files describe *cmds* and *args* to *fcntl*(2) and *open*(2).

```
/* Flag values accessible to open(2) and fcntl(2) */
/* (The first three can only be set by open) */
#define O_RDONLY 0
#define O_WRONLY 1
#define O_RDWR   2
#define O_NDELAY 04        /* Nonblocking I/O */
#define O_APPEND 010       /* append (writes guaranteed at the end) */
#define O_SYNC   020       /* synchronous write option */

/* Flag values accessible only to open(2) */
#define O_CREAT  00400     /* open w/file create (uses third open arg) */
#define O_TRUNC  01000     /* open with truncation */
#define O_EXCL   02000     /* exclusive open */

/* Flag values accessible only to fcntl(2) */
#define FASYNC   0x8000    /* Enable the SIGIO signal */

/* fcntl(2) requests */
#define F_DUPFD   0        /* Duplicate fildes */
#define F_GETFD   1        /* Get fildes flags */
#define F_SETFD   2        /* Set fildes flags */
#define F_GETFL   3        /* Get file flags */
#define F_SETFL   4        /* Set file flags */
#define F_GETLK   5        /* Get file lock */
#define F_SETLK   6        /* Set file lock */
#define F_SETLKW  7        /* Set file lock and wait */
#define F_CHKFL   8        /* Check legality of file flag changes */
#define F_SETOWN  126      /* Set to receive signals */
#define F_GETOWN  127      /* Get ID of signal receiver */

/* file segment locking control structure */
struct flock {
        short 1_type;
        short 1_whence;
        long  1_start;
        long  1_len;       /* if 0 then until EOF */
        short 1_sysid;     /* returned with F_GETLK */
        short 1_pid;        /* returned with F_GETLK */
};

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

SEE ALSO

fcntl(2).
open(2) in the *UNIX System V Programmer's Reference Manual*

NAME

stat - data returned by stat system call

SYNOPSIS

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

DESCRIPTION

stat(2) and *fstat*(2) return data whose structure is defined by the **<sys/stat.h>** include file. The encoding of the field *st_mode* is defined in this file also.

Structure of the result of *stat*(2) is as follows:

```
struct stat {
        dev_t     st_dev;
        ushort    st_ino;
        ushort    st_mode;
        short     st_nlink;
        ushort    st_uid;
        ushort    st_gid;
        dev_t     st_rdev;
        off_t     st_size;
        time_t    st_atime;
        time_t    st_mtime;
        time_t    st_ctime;
};

#define S_IFMT    0170000 /* type of file */
#define S_IFDIR   0040000 /* directory */
#define S_IFCHR   0020000 /* character special */
#define S_IFBLK   0060000 /* block special */
#define S_IFREG   0100000 /* regular */
#define S_IFIFO   0010000 /* fifo */
#define S_IFSOCK  0070000 /* socket */
#define S_ISUID   04000   /* set user ID on execution */
#define S_ISGID   02000   /* set group ID on execution */
#define S_ISVTX   01000   /* save swapped text even after use */
#define S_IREAD   00400   /* read permission, owner */
#define S_IWRITE  00200   /* write permission, owner */
#define S_IEXEC   00100   /* execute/search permission, owner */
#define S_ENFMT   S_ISGID /* record locking enforcement flag */
#define S_IRWXU   00700   /* read,write, execute: owner */
#define S_IRUSR   00400   /* read permission: owner */
#define S_IWUSR   00200   /* write permission: owner */
#define S_IXUSR   00100   /* execute permission: owner */
#define S_IRWXG   00070   /* read, write, execute: group */
```

```
#define S_IRGRP  00040    /* read permission: group */
#define S_IWGRP  00020    /* write permission: group */
#define S_IXGRP  00010    /* execute permission: group */
#define S_IRWXO  00007    /* read, write, execute: other */
#define S_IROTH  00004    /* read permission: other */
#define S_IWOTH  00002    /* write permission: other */
#define S_IXOTH  00001    /* execute permission: other */
```

SEE ALSO

types(5).
stat(2) in the *UNIX System V Programmer's Reference Manual.*

NAME

types - primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

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

```
typedef  struct { int r[1]; } * physadr;
typedef  long            daddr_t;       /* <disk address> type */
typedef  char *          caddr_t;       /* <core address> type */
typedef  unsigned char   unchar;
typedef  unsigned short  ushort;
typedef  unsigned int    uint;
typedef  unsigned long   ulong;
typedef  ushort          ino_t;         /* <inode> type */
typedef  short           cnt_t;         /* <count> type */
typedef  long            time_t;        /* <time> type */
typedef  int             label_t[44];
typedef  short           dev_t;         /* <old device number> type */
typedef  long            off_t;         /* <offset> type */
typedef  long            paddr_t;       /* <physical address> type */
typedef  int             key_t;         /* IPC key type */
typedef  unsigned char   use_t;         /* use count for swap.  */
typedef  short           sysid_t;
typedef  short           index_t;
typedef  short           lock_t;        /* lock work for busy wait */
typedef  unsigned int    size_t;        /* len param for string funcs */
typedef  unsigned char   u_char;
typedef  unsigned short  u_short;
typedef  unsigned int    u_int;
typedef  unsigned long   u_long;
typedef  u_short         uid_t;
typedef  u_short         gid_t;
typedef  struct _quad { long val[2]; } quad;
typedef  long            swblk_t;

#define NBBY             8              /* number of bits per byte */
#define FD_SETSIZE       256
#define NFDBITS (sizeof(long) * NBBY)   /* bits per mask */
#ifndef howmany
#define howmany(x, y)  (((x)+((y)-1))/(y))
#endif

typedef struct fd_set {
        long fds_bits[howmany(FD_SETSIZE, NFDBITS)];
} fd_set;
```

```
#define FD_SET(n, p)      ((p)->fds_bits[(n)/NFDBITS] |= (1 << ((n) % NFDBITS)))
#define FD_CLR(n, p)      ((p)->fds_bits[(n)/NFDBITS] &= ~(1 << ((n) % NFDBITS)))
#define FD_ISSET(n, p)    ((p)->fds_bits[(n)/NFDBITS] & (1 << ((n) % NFDBITS)))
#define FD_ZERO(p)        bzero((char *)(p), sizeof(*(p)))
```

The form *daddr_t* is used for disk addresses except in an i-node on disk, see *fs*(4). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The *label_t* variables save the processor state while another process is running.

NAME

values - machine-dependent values

SYNOPSIS

include <values.h>

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is the value of the high-order bit.

BITS(*type*)	The number of bits in a specified *type* (e.g., int).
HIBITS	The value of a short integer with only the high-order bit set (in most implementations, 0x8000).
HIBITL	The value of a long integer with only the high-order bit set (in most implementations, 0x80000000).
HIBITI	The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).
MAXSHORT	The maximum value of a signed short integer (in most implementations, 0x7FFF ≡ 32767).
MAXLONG	The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF ≡ 2147483647).
MAXINT	The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).
MAXFLOAT	The maximum value of a single-precision floating-point number.
MAXDOUBLE, LN_MAXDOUBLE	The maximum value of a double-precision floating-point number and its natural logarithm.
MINFLOAT	The minimum positive value of a single-precision floating-point number.
MINDOUBLE, LN_MINDOUBLE	The minimum positive value of a double-precision floating-point number and its natural logarithm.
FSIGNIF	The number of significant bits in the mantissa of a single-precision floating-point number.
DSIGNIF	The number of significant bits in the mantissa of a double-precision floating-point number.

SEE ALSO

intro(3).
math(5) in the *UNIX System V Programmer's Reference Manual.*