

### A1.1 Introductory reading

If you are unfamiliar with the UNIX operating system, you may wish to do some introductory reading, for example:

Banahan and Rutter, *UNIX, The Book*, Zigma Technical Press

Bourne, S.R., *The UNIX System*, Addison-Wesley, ISBN 0-201-13791-7

Gauthier, R., *Using the UNIX System*, Reston Publishing Company Inc (A Prentice-Hall Company), ISBN 0-8359-8164-9

Thomas, *A User Guide to the UNIX System*, Osborne/McGraw-Hill, ISBN 0-931988-71-3

Lumoto, A. N. and Lomuto, N., *A UNIX Primer*, Prentice-Hall, Englewood Cliffs, New Jersey

Bourne has also written articles which appear in 'The Blue Book' (see below) and the *UPM* Volume 2A. Some parts of his book repeat this information slightly more comprehensively. Bourne's book covers a lot of ground but seems very short of real introductory level material. It is probably more useful once you have a rough idea as to what UNIX is all about. The Banahan and Rutter book is a good introduction to Bourne.

Gauthier's book is extremely simple, perhaps too simple, and certainly not a complete introduction. The Lomutos' book is very recent and not particularly relevant to PNX but it does provide a good introduction to using *nroff*.

The *UPM* contains several introductory articles.

Part 2 of the Bell System Technical Journal issue for July-August 1978 (that is, volume 57, number 6, part 2: ISSN0005-8580) was devoted to UNIX and related matters. Reprints of this document, which is often referred to in UNIX circles as 'The Blue Book', are available through bookshops. It contains:

Preface

Foreword

The UNIX Time-Sharing System

UNIX Implementation

A Retrospective

The UNIX Shell

The C Programming Language

Portability of C Programs and the UNIX System

The MERT Operating System

UNIX on a Microprocessor

A Minicomputer Satellite Processor System

Document Preparation

Statistical Text Processing

Language Development Tools

The Programmer's Workbench

The UNIX Operating System as a Base for Applications

Microcomputer Control of Apparatus, Machinery, and Experiments

Circuit Design Aids

A Support Environment for MAC-8 Systems

No. 4 ESS Diagnostic Environment

RBCS/RCMAS - Converting to the MERT Operating System

The Network Operations Center System

Whilst many of these articles are partly or wholly irrelevant to PNX on the ICL PERQ, some of the early articles are useful introductions to the structure of the UNIX operating system.

When you have access to PNX on PERQ, you may find the *learn(1)* command useful; it provides computer-aided instruction courses and practice in the use of the UNIX operating system, and is therefore mostly applicable to PNX.

UNIX is mostly written in the C programming language and many of the features of UNIX reflect the features of C. Before trying to understand the way UNIX does things it would be useful to gain some familiarity with the C programming language:

*The C Programming Language*, Kernighan and Ritchie, Prentice-Hall, Inc, Englewood Cliffs, New Jersey 07632

See also the articles in 'The Blue Book' and in the *UPM*.

#### *A suggested plan of attack*

The UNIX Time Sharing System                   in *UPM Volume 2A* and 'The Blue Book'

UNIX Implementation                           in 'The Blue Book'

A Retrospective                                in 'The Blue Book'

The UNIX Shell                                in 'The Blue Book'

The 'C' Programming Language                in 'The Blue Book'

The UNIX System                              Bourne S R

The 'C' Programming Language                Kernighan and Ritchie

These last two are good general reference texts.

### A1.2 **The UNIX Programmers Manual**

The *UPM* is available through ICL (see Preface) in exactly the same form as that published by Bell Laboratories, so that:

- 1     The *UPM* is useful to you if you are also working on a UNIX implementation other than *PNX*
- 2     A new issue can be made available more quickly in the event of changes by the publisher

However, this means that some items in the *UPM* are inaccurate for, or irrelevant to, *PNX*. Appendix 2 defines the relevance to *PNX* of each item in Volume 1 of the *UPM* and Appendix 3 contains the necessary extra specifications in a style that makes it easy to add sheets to the *UPM* if you wish.

The *UPM* consists of two volumes in three binders. *Volume 1* mainly contains specification sheets for commands, system calls, subroutines, and files. *Volume 2* (supplied for reasons of size as volumes 2A and 2B) contains 38 papers on subjects of interest to trainee, general, and specialist users of the UNIX operating system.

#### A1.2.1 **UPM volume 1**

Volume 1 of the *UPM* is divided into eight sections:

- 1     Commands
- 2     System calls
- 3     Subroutines
- 4     Special files
- 5     File formats and conventions
- 6     Games
- 7     Macro packages and language conventions
- 8     Maintenance

All entries in the *UPM Volume 1* are headed by the appropriate name in capitals followed by the section number in brackets, for example, *ed(1)*, *filsys(5)*. Such references elsewhere in this *Guide to PNX* are to *Volume 1* of the *UPM* or additional specifications in Appendix 3. Note that sometimes the number has a letter appended to indicate the type of item, for example *clri(1M)*, where M indicates that this is a maintenance command.

All references in *UPM Volume 1* to assembler are to PDP assembler, and are therefore irrelevant to *PNX* on the ICL PERQ.

A list of all entries in the *UPM Volume 1* is given in Appendix 2 of this *Guide to PNX*.

Note that *UPM Volume 1* is online and pages can be accessed using the *man(1)* command followed by the name of the specification, for example:

**Man ls**

### A1.2.2 UPM Volume 2

*Volume 2* consists of a collection of articles on UNIX. Its contents list is given at the front of *Volume 2A*; a shortened copy of the contents is repeated here, accompanied by some comments on the relevance of each article to PNX. *Volume 2A* contains:

No	Title	Comments
1	7th Edition UNIX - Summary	<i>What's new</i> : section A is correct but a bit irrelevant if you have no experience of the 6th Edition: section B (hardware) is irrelevant; section C (software) is a useful categorisation
2	The UNIX Time-Sharing System	Interesting and mostly appropriate
3	UNIX For Beginners	May be worth working through if you are a beginner
4	A Tutorial Introduction to the UNIX Text Editor	Relevant if you want to use <i>ed(1)</i> but PNX provides <i>spy(1)</i> (see Chapter 4 of this publication)
5	Advanced Editing on UNIX	Relevant if you are using <i>ed(1)</i>
6	An Introduction to the UNIX SHELL	Important for gaining an understanding of the command language
7	Learn - Computer Aided Instruction on UNIX	May be useful if you are a beginner
8	Typing Documents on the UNIX System	Relevant
9	A System for Typesetting Mathematics	Only relevant if you plan to connect appropriate phototypesetting equipment
10	TBL - A program to Format Tables	As 9
11	Some Applications of Inverted Indexes on the UNIX System	Relevant
12	NROFF/TROFF User's Manual	<i>nroff</i> relevant
13	A TROFF Tutorial	As 12
14	The C Programming Language - Reference Manual	See Chapter 8 of this <i>Guide to PNX</i>
15	Lint, A C Program Checker	As 14
16	Make - A Program for Maintaining Computer Programs	Relevant
17	UNIX Programming	Relevant
18	A Tutorial Introduction to ADB	<i>adb</i> is not available under PNX

*Volume 2B* contains:

No	Title	Comments
19	YACC: Yet Another Compiler Compiler	Relevant
20	LEX - A Lexical Analyzer Generator	Relevant
21	A Portable Fortran 77 Compiler	You may find more useful the ICL publication <i>FORTRAN 77 Language</i> , together with Appendix 4 to this <i>Guide to PNX</i>
22	Ratfor - A Preprocessor for a Rational Fortran	Relevant
23	The M4 Macro Processor	Not supported under PNX
24	SED A Non-interactive Text Editor	Relevant
25	AWK - A pattern Scanning and Processing Language	Relevant
26	DC - An interactive Desk Calculator	Relevant
27	BC - An Arbitrary Precision Desk-Calculator Language	Relevant
28	UNIX Assembler Reference Manual	Irrelevant to PERQ since this article refers to PDP assembler
29	Setting up UNIX	Irrelevant; see instead Chapter 2 of this <i>Guide to PNX</i>
30	Regenerating System Software	Irrelevant
31	UNIX Implementation	Partly relevant
32	The UNIX I/O System	Relevant
33	A Tour Through the UNIX C Compiler	Irrelevant since this is not the compiler implemented under PNX; see article 34
34	A Tour Through the Portable C Compiler	Relevant
35	A Dial-Up Network of UNIX Systems	Relevant
36	UUCP Implementation Description	Relevant
37	On the Security of UNIX	Not as relevant to the single workstation PERQ as to a multi-terminal UNIX system
38	Password Security: A Case History	Relevant, though less important since with PERQ it may be convenient to add physical access control to password access control

PNX comprises three sets of components:

- 1 Some but not all of the standard Version 7 UNIX facilities
- 2 Some System 3 UNIX facilities
- 3 PNX versions of UNIX facilities and additions to the standard facilities

The standard Version 7 components are formally specified in the eight sections of the *UNIX Programmer's Manual, Volume 1*. The System 3 and PNX additions are specified in Appendix 3.

The specifications in Appendix 3 are designed to be inserted into the *UPM, Volume 1*. This appendix is a contents list designed to be inserted in the beginning of the *UPM, Volume 1* before the permuted index. The contents list includes all the Version 7 UNIX specifications and the System 3 UNIX and PNX additions. A column in the contents list indicates if a particular component is Version 7 UNIX, System 3 UNIX or PNX. A further column indicates the following:

- 1 A + indicates the Version 7 item is supported
- 2 A X indicates the Version 7 item is not supported. Cross out these items in your *UPM, Volume 1*. If possible remove the pages
- 3 An \* indicates the PNX or System 3 component replaces an existing Version 7 component. In this case you should cross out the Version 7 specification from the *UPM, Volume 1* and insert the specifications in Appendix 3 as near as possible to the crossed out pages. If possible remove and replace
- 4 No entry in this column indicates that the component is an addition. Insert the specifications in the order suggested by this appendix or as near as possible to this order

## CONTENTS OF PNX Release 2.0

## 1 Commands and application programs

intro	introduction to commands and applications programs	UNIX 7	✓
ac	login accounting	UNIX 7	✓
adb	debugger	UNIX 7	X
admin	create and administer SCCS files	UNIX 3	
ar	archive and library maintainer	UNIX 7	✓
arcv	converts archives to new format	UNIX 7	X
as	assembler (replaced)	UNIX 7	X
as	PNX assembler	PNX	*
at	execute commands at a later time	UNIX 7	✓
awk	pattern scanning and processing language	UNIX 7	✓
banner	makes posters	UNIX 3	
bas	BASIC language	UNIX 7	X
basename	strip filename affixes	UNIX 7	✓
bc	arbitrary precision arithmetic language	UNIX 7	✓
bdiff	compares large files to find differences	UNIX 3	
cal	print calendar	UNIX 7	✓
calendar	reminder service	UNIX 7	✓
cat	catenate and print	UNIX 7	✓
cb	C program beautifier	UNIX 7	✓
cc	C compiler (replaced)	UNIX 7	X
cc	C compiler	PNX	*
cd	change working directory	UNIX 7	✓
cdc	change the delta commentary of an SCCS delta	UNIX 3	
cedra	screen oriented interactive cursor editor	PNX	
chatter	teletype emulation	PNX	
chmod	change mode	UNIX 7	✓
chown	change ownership	UNIX 7	✓
chroot	change root directory	UNIX 3	
clri	clear i-node	UNIX 7	✓
cmp	compare two files	UNIX 7	✓
col	filter reverse line feeds	UNIX 7	✓
comb	combine SCCS deltas	UNIX 3	
comm	select or reject lines common to two sorted files	UNIX 7	✓
cp	copy	UNIX 7	✓
cpio	copy file archives in and out	UNIX 3	
cptree	filestore subtree copy	PNX	
cref	make cross reference listing	UNIX 3	

<b>crypt</b>	encode/decode	UNIX 7	✓
<b>csplit</b>	context split	UNIX 3	
<b>cu</b>	call UNIX	UNIX 7	X
<b>cut</b>	cut out selected fields of each line of a file	UNIX 3	
<b>date</b>	print and set the date	UNIX 7	✓
<b>dc</b>	desk calculator	UNIX 7	✓
<b>dcheck</b>	file system directory consistency check	UNIX 7	✓
<b>dd</b>	convert and copy a file	UNIX 7	✓
<b>debugoff</b>	disable or enable the microcode debugger (including <i>debugon</i> )	PNX	
<b>delta</b>	make a delta (change) to an SCCS file	UNIX 3	
<b>deroff</b>	remove <i>nroff</i> , <i>troff</i> , <i>tbl</i> and <i>eqn</i> constructs	UNIX 7	✓
<b>df</b>	disc free	UNIX 7	✓
<b>diff</b>	differential file comparator	UNIX 7	✓
<b>diff3</b>	3-way differential file comparator	UNIX 7	✓
<b>diffmk</b>	mark differences between files	UNIX 3	
<b>dircomp</b>	directory comparison	UNIX 3	
<b>du</b>	summarise disc usage	UNIX 7	✓
<b>dump</b>	incremental file system dump	UNIX 7	X
<b>dumpdir</b>	print the names of files on a dump tape	UNIX 7	X
<b>echo</b>	echo arguments	UNIX 7	✓
<b>ed</b>	text editor	UNIX 7	✓
<b>enroll</b>	secret mail (including <i>xsend</i> , <i>xget</i> )	UNIX 7	X
<b>env</b>	set environment for command execution	UNIX 3	
<b>eqn</b>	typeset mathematics (including <i>neqn</i> , <i>checkeq</i> )	UNIX 7	✓
<b>expr</b>	evaluate arguments as an expression	UNIX 7	✓
<b>f77</b>	FORTRAN 77 compiler (replaced)	UNIX 7	X
<b>f77</b>	PNX FORTRAN 77 compiler	PNX	*
<b>factor</b>	factor a number, generate large primes (including <i>primes</i> )	UNIX 7	X
<b>file</b>	determine file type	UNIX 7	✓
<b>find</b>	find files	UNIX 7	✓
<b>fl</b>	floppy format and transfer utility	PNX	
<b>fsck</b>	file system consistency check and interactive repair	UNIX 3	
<b>get</b>	get a version of an SCCS file	UNIX 3	
<b> getopt</b>	parse command options	UNIX 3	
<b>graph</b>	draw a graph	UNIX 7	X
<b>grep</b>	search a file for a pattern (including <i>egrep</i> , <i>fgrep</i> )	UNIX 7	✓
<b>help</b>	ask for help	UNIX 3	
<b>hyphen</b>	find hyphenated words	UNIX 3	
<b>icheck</b>	file system storage consistency check	UNIX 7	✓
<b>id</b>	print user and group IDs and names	UNIX 3	

install	install commands	UNIX 3
iostate	report I/O statistics	UNIX 7 ✓
join	relational database operator	UNIX 7 ✓
kill	terminates or signals a process	UNIX 7 ✓
ld	loader (replaced)	UNIX 7 X
ld	PNX loader	PNX *
learn	computer aided instruction about UNIX	UNIX 7 ✓
lex	generator of lexical analysis programs	UNIX 7 ✓
line	read one line	UNIX 3
lint	C program verifier	UNIX 7 ✓
link	exercise link and unlink system calls	UNIX 3
ln	make a link	UNIX 7 ✓
login	sign on	UNIX 7 ✓
look	find lines in a sorted list	UNIX 7 ✓
lookall	look through all text files on UNIX	UNIX 7 X
lorder	find ordering relation for an object library	UNIX 7 ✓
lpr	line printer spooler (replaced)	UNIX 7 X
lpr	PNX spooling system (including <i>lpc</i> , <i>lpp</i> )	PNX *
ls	list contents of directory	UNIX 7 ✓
lstree	list the contents of any subtree of the PNX file system	PNX
m4	macro processor	UNIX 7 X
mail	send or receive mail among users	UNIX 7 ✓
make	maintain program groups	UNIX 7 ✓
makepxn	establish a PNX system on the fixed disc (including <i>updatepxn</i> )	PNX
man	print sections of the manual on the screen	UNIX 7 ✓
mesg	permit or deny messages	UNIX 7 ✓
mftp	PNX Open Systems LAN file transfer program	PNX
mkconf	generate configuration tables	UNIX 7 X
mkdir	make a directory	UNIX 7 ✓
mkflop	create an empty PNX file store on a floppy disc	PNX
mkfs	construct a file system (replaced)	UNIX 7 X
mkfs	construct a PNX file system	PNX *
mknet	set up a transport address file	PNX
mknod	build a special file	UNIX 7 ✓
mkwind	make a window description file	PNX
mount	mount and dismount file system (including <i>umount</i> )	UNIX 7 ✓
mountflop	mount and dismount file system on the floppy disc drive (including <i>umountflop</i> )	PNX
mv	move or rename files and directories	UNIX 7 ✓
mvdir	move a directory	UNIX 3

ncheck	generate names from i-numbers	UNIX 7	✓
newgrp	login to a new group	UNIX 7	✓
news	print news items	UNIX 3	
nice	run a command at low priority (including <i>nohup</i> )	UNIX 7	✓
nl	line numbering filter	UNIX 3	
nm	print name list (replaced)	UNIX 7	X
nm	print name list	PNX	*
od	octal dump	UNIX 7	✓
pack	compress and expand files (including <i>pcat</i> , <i>unpack</i> )	UNIX 3	
passwd	change login password	UNIX 7	✓
paste	merge same lines of several files or subsequent lines of one file	UNIX 3	
plot	graphics filters (replaced)	UNIX 7	X
plot	plot screen or window to spooler (including <i>splot</i> )	PNX	*
pr	print file	UNIX 7	✓
prep	prepare text for statistical processing	UNIX 7	X
prime	copy filestore issue disc to fixed disc	PNX	
printerr	error reporting daemon	PNX	
prof	display profile data	UNIX 7	X
prs	print an SCCS file	UNIX 3	
ps	process status	UNIX 7	✓
pstat	print system facts	UNIX 7	✓
ptx	permuted index	UNIX 7	✓
pubindex	make inverted bibliographic index	UNIX 7	X
pwck	password file and group file checkers (including <i>grpck</i> )	UNIX 3	
pwd	print working directory name	UNIX 7	✓
quot	summarise file system ownership	UNIX 7	X
ranlib	update symbol definitions for archive files	PNX	
ratfor	rational FORTRAN dialect	UNIX 7	✓
refer	find and insert literature references in documents (including <i>lookbib</i> )	UNIX 7	✓
regcmp	regular expression compile	UNIX 3	
restor	incremental file system restore	UNIX 7	X
rev	reverse lines of a file	UNIX 7	✓
rm	remove, unlink files (including <i>rmdir</i> )	UNIX 7	✓
rmdel	remove a delta from an SCCS file	UNIX 3	
roff	format text (see <i>troff</i> )	UNIX 7	X
rstty	create a process group to run a control terminal on the RS232C interface	PNX	
sa	system accounting (including <i>accton</i> )	UNIX 7	✓
sact	print current SCCS file editing activities	UNIX 3	

sccsdiff	compare two versions of an SCCS file	UNIX 3
sdb	symbolic debugger	PNX
sed	stream editor	UNIX 7 ✓
setpf	sets/returns strings assigned to the function keys	PNX
sh	command language interpreter	UNIX 7 ✓
shutdown	terminate all processing	PNX
size	size of an object file	UNIX 7 ✓
slaveft	responds to <i>mftp</i> or PERQFTP	PNX
sleep	suspend execution for an interval	UNIX 7 ✓
sort	sort or merge files	UNIX 7 ✓
spell	find spelling errors (including <i>spellin</i> , <i>spellout</i> )	UNIX 7 ✓
spline	interpolate smooth curve	UNIX 7 X
split	split a file into pieces	UNIX 7 ✓
spy	screen oriented interactive text editor	PNX
srs	set RS232C device options	PNX
strip	remove symbols and relocation bits	UNIX 7 ✓
struct	structure FORTRAN programs	UNIX 7 ✓
stty	set terminal options (replaced)	UNIX 7 X
stty	set PNX terminal options	PNX *
su	substitute user ID temporarily	UNIX 7 ✓
sum	sum and count blocks in a file	UNIX 7 ✓
sync	update the superblock	UNIX 7 ✓
tabs	set terminal tabs (irrelevant)	UNIX 7 X
tail	deliver the last part of the file	UNIX 7 ✓
tar	tape archiver	UNIX 7 ✓
tbl	format tables for nroff and troff	UNIX 7 ✓
tc	phototypesetter emulator	UNIX 7 X
tee	pipe fitting	UNIX 7 ✓
test	condition demand	UNIX 7 ✓
time	time a command	UNIX 7 ✓
tk	paginator for Tektronix 4014	UNIX 7 X
touch	update date last modified of a file	UNIX 7 ✓
tp	manipulate tape archive	UNIX 7 X
tr	translate characters	UNIX 7 ✓
troff	text formatting and typesetting- <i>nroff</i> is available	UNIX 7 <i>nroff</i> only
true	provide truth values (including false)	UNIX 7 ✓
tsort	topological sort	UNIX 7 ✓
tty	get terminal name	UNIX 7 ✓
uname	print name of current UNIX	UNIX 3
unget	undo a previous get of an SCCS file	UNIX 3

uniq	report repeated lines in a file	UNIX 7	✓
units	conversion program	UNIX 7	X
uucp	UNIX to UNIX copy (including <i>uulog</i> )	UNIX 7	
uux	UNIX to UNIX command execution	UNIX 7	X
val	validate an SCCS file	UNIX 3	
wait	await completion of a process	UNIX 7	✓
wall	write to all users	UNIX 7	✓
wc	word count	UNIX 7	✓
what	identify SCCS files	UNIX 3	
who	who is on the system	UNIX 7	✓
whodo	who is doing what	UNIX 3	
winit	initialise the Window Management System	PNX	
write	write to another user	UNIX 7	✓
wsetup	make standard WMS window description files	PNX	
xargs	construct argument list(s) and execute command	UNIX 3	
xref	cross reference for C programs	UNIX 3	
yacc	yet another compiler compiler	UNIX 7	✓

## 2 Systems calls

intro	introduction to system calls and error numbers	UNIX 7	✓
access	determines accessibility of file	UNIX 7	✓
acct	turn accounting on or off	UNIX 7	✓
alarm	schedule signal after specified time	UNIX 7	✓
brk	change core allocation (including <i>sbrk</i> , <i>break</i> )	UNIX 7	✓
chdir	change default directory (including <i>chroot</i> )	UNIX 7	✓
chmod	change mode of file	UNIX 7	✓
chown	change owner and group of a file	UNIX 7	✓
close	close a file	UNIX 7	✓
creat	create a new file	UNIX 7	✓
dup	duplicate an open file description (including <i>dup2</i> )	UNIX 7	✓
exec	execute file; <i>exec1</i> , <i>execv</i> , <i>execle</i> , <i>execve</i> , <i>execlp</i> , <i>execup</i> , <i>exec</i> , <i>exece</i> , <i>environ</i>	UNIX 7	✓
exit	terminate process	UNIX 7	✓
fork	spawn new process	UNIX 7	✓
getuid	get user and group identity; <i>getuid</i> , <i>getgid</i> , <i>geteuid</i> , <i>getegid</i>	UNIX 7	✓
getpid	get process identification	UNIX 7	✓
indir	indirect system call	UNIX 7	X
ioctl	control device (replaced)	UNIX 7	X
ioctl	control device (including <i>stty</i> , <i>gtty</i> )	PNX	*
kill	send signal to a process	UNIX 7	✓
link	link to a file	UNIX 7	✓

lock	lock a process in primary memory	UNIX 7	✓
lseek	move <i>read/write</i> pointer (including <i>tell</i> )	UNIX 7	✓
mknod	make a PNX directory or a special file (replaced)	UNIX 7	X
mknod	make a PNX directory or a special file	PNX	*
mount	mount or remove a file system (including <i>umountfl</i> )	UNIX 7	✓
mpx	create and manipulate multiplexed files	UNIX 7	X
nice	get program priority	UNIX 7	✓
open	open for reading or writing	UNIX 7	✓
pause	stop until signal	UNIX 7	✓
phys	allow a process to access physical address	UNIX 7	X
pipe	create an interprocess channel	UNIX 7	✓
pkon	establish packet protocol (including <i>pkoff</i> )	UNIX 7	X
profil	execution time profile	UNIX 7	X
ptrace	process trace (replaced)	UNIX 7	X
ptrace	process trace	PNX	*
read	read from file	UNIX 7	✓
setuid	set user and group ID (including <i>setgid</i> )	UNIX 7	✓
signal	catch or ignore signals (replaced)	UNIX 7	X
signal	PNX signals to catch or ignore	PNX*	
stat	get file status (including <i>fstat</i> )	UNIX 7	✓
stime	set time	UNIX 7	✓
sync	update superblock	UNIX 7	✓
time	get date and time (including <i>ftime</i> )	UNIX 7	✓
times	get process times	UNIX 7	✓
umask	set file creation mode mask	UNIX 7	✓
unlink	remove directory entry	UNIX 7	✓
utime	set file times	UNIX 7	✓
wait	wait for process to terminate	UNIX 7	✓
wdbyte	draw characters from a string to a window	PNX	
wgread	read window graphics input from tablet	PNX	
wline	draw a line on a window	PNX	
wrasop	window raster operation	PNX	
write	write on a file	UNIX 7	✓

### 3 Subroutines

intro	introduction to library functions	UNIX 7	✓
abort	generate IOT fault	UNIX 7	✓
abs	integer absolute value	UNIX 7	✓
assert	program verification	UNIX 7	✓
atof	convert ASCII to numbers (including <i>atoi</i> , <i>atol</i> )	UNIX 7	✓
build	text build an array of strings from a menu file	UNIX 7	✓

<b>crypt</b>	DES encryption (including <i>setkey</i> , <i>encrypt</i> )	UNIX 7	✓
<b>ctime</b>	convert date and time to ASCII (including <i>localtime</i> , <i>gmtime</i> , <i>asctime</i> , <i>timezone</i> )	UNIX 7	✓
<b>ctype</b>	character classification; <i>isalpha</i> , <i>isupper</i> , <i>islower</i> , <i>isdigit</i> , <i>isalnum</i> , <i>isspace</i> , <i>ispunct</i> , <i>ispprint</i> , <i>iscntrl</i> , <i>isascii</i>	UNIX 7	✓
<b>ctype</b>	as above and; <i>isxdigit</i> , <i>isgraph</i>	UNIX 3	
<b>dbm</b>	data base subroutines; <i>dbminit</i> , <i>fetch</i> , <i>store</i> , <i>delete</i> , <i>firstkey</i> , <i>nextkey</i>	UNIX 7	✓
<b>ecvt</b>	output conversion (including <i>fcvt</i> , <i>gcvt</i> )	UNIX 7	✓
<b>end</b>	last locations in program (including <i>etext</i> , <i>edata</i> )	UNIX 7	✓
<b>exp</b>	exponential, logarithm, power, squareroot; <i>exp</i> , <i>log</i> , <i>log10</i> , <i>pow</i> , <i>sqr</i>	UNIX 7	✓
<b>fclose</b>	close or flush a stream (including <i>fflush</i> )	UNIX 7	✓
<b>ferror</b>	stream, status enquires (including <i>feof</i> , <i>clearerr</i> , <i>fileno</i> )	UNIX 7	✓
<b>floor</b>	absolute value, floor, ceiling functions (including <i>fabs</i> , <i>ceil</i> )	UNIX 7	✓
<b>fopen</b>	open a stream (including <i>freopen</i> , <i>fdopen</i> )	UNIX 7	✓
<b>fread</b>	buffered binary I/O (including <i>fwrite</i> )	UNIX 7	✓
<b>frexp</b>	split into mantissa and exponent (including <i>ldexp</i> , <i>modf</i> )	UNIX 7	✓
<b>fseek</b>	reposition a stream (including <i>ftell</i> , <i>rewind</i> )	UNIX 7	✓
<b>getc</b>	get character or word from a stream (including <i>getchar</i> , <i>fgetc</i> , <i>getw</i> )	UNIX 7	✓
<b>getenv</b>	value for environment name	UNIX 7	✓
<b>getgrent</b>	get group file entry (including <i>getgrgid</i> , <i>getgrnam</i> , <i>setgrent</i> , <i>endgrent</i> )	UNIX 7	✓
<b>getlogin</b>	get login name	UNIX 7	✓
<b>getopt</b>	get option letter from argv	UNIX 3	
<b>getpass</b>	read a password	UNIX 7	✓
<b>getpw</b>	get name from UID	UNIX 7	✓
<b>getpwent</b>	get password file entry (including <i>getpwuid</i> , <i>getpwnam</i> , <i>setpwent</i> , <i>endpwent</i> )	UNIX 7	✓
<b>gets</b>	get a string from a stream (including <i>fgets</i> )	UNIX 7	✓
<b>hypot</b>	euclidean distance (including <i>cabs</i> )	UNIX 7	✓
<b>jo</b>	bessel function's (including <i>j1</i> , <i>jn</i> , <i>y0</i> , <i>y1</i> , <i>yn</i> )	UNIX 7	✓
<b>l3tol</b>	convert between 3 byte integers and long integers (including <i>ltol3</i> )	UNIX 7	✓
<b>lplota</b>	copy window to an area of memory	PNX	
<b>lplotb</b>	copy window image to a file	PNX	
<b>lpq</b>	adds file to spool queue (including <i>lpdq</i> )	PNX	
<b>malloc</b>	main memory allocator (including <i>free</i> , <i>realloc</i> , <i>calloc</i> )	UNIX 7	✓
<b>menuesc</b>	routine to force escape from <i>pickmenu</i>	PNX	
<b>mktemp</b>	make a unique filename	UNIX 7	✓
<b>monitor</b>	prepare execution profile	UNIX 7	✓

mp	multiple precision integer arithmetic: <i>itom</i> , <i>madd</i> , <i>msub</i> , <i>sult</i> , <i>mdic</i> , <i>min</i> , <i>mout</i> , <i>pow</i> , <i>gcd</i> , <i>rpow</i>	UNIX 7	✓
nlist	get entries from name list	UNIX 7	✓
perror	system error messages (including <i>sys errlist</i> , <i>sys nerr</i> )	UNIX 7	✓
pickmenu	pop up menu package	PNX	
pkopen	packet driver simulator	UNIX 7	X
plot	graphics interface	UNIX 7	X
popen	initiate I/O to or from a process (including <i>pclose</i> )	UNIX 7	✓
printf	formatted output conversion (including <i>sprintf</i> , <i>sprintf</i> )	UNIX 7	✓
putc	put character on a word or a stream (including <i>putchar</i> , <i>fputc</i> , <i>putw</i> )	UNIX 7	✓
puts	put a string on a stream (including <i>fputs</i> )	UNIX 7	✓
qsort	quicker sort	UNIX 7	✓
rand	random number generator (including <i>rand</i> )	UNIX 7	✓
rectangle	allocate memory suitable for graphics operations	PNX	
regex	regular expression compile/execute (including <i>regcmp</i> )	UNIX 3	
scanf	formatted input conversion (including <i>fscanf</i> , <i>sscanf</i> )	UNIX 7	✓
setbuf	assign buffering to a stream	UNIX 7	✓
setjmp	non-local goto (including <i>longjmp</i> )	UNIX 7	✓
sin	trigonometric functions (including <i>cos</i> , <i>tan</i> , <i>asin</i> , <i>acos</i> , <i>atan</i> , <i>atan2</i> )	UNIX 7	✓
sinh	hyperbolic functions (including <i>cosh</i> , <i>tanh</i> )	UNIX 7	✓
sleep	suspend execution for an interval	UNIX 7	✓
stdio	standard buffered I/O	UNIX 7	✓
string	string operations; <i>strcat</i> , <i>strncat</i> , <i>strcmp</i> , <i>strncmp</i> , <i>strcpy</i> , <i>strncpy</i> , <i>strlen</i> , <i>index</i> , <i>rindex</i>	UNIX 7	✓
string	string operations: <i>strcat</i> , <i>strncat</i> , <i>strcmp</i> , <i>strncmp</i> , <i>strcpy</i> , <i>strncpy</i> , <i>strlen</i> , <i>strchr</i> , <i>strrchr</i> , <i>strpbrk</i> , <i>strspn</i> , <i>strcspn</i> , <i>strtok</i>	UNIX 3	
swab	swap bytes	UNIX 7	✓
system	issue a shell command	UNIX 7	✓
ttyname	find name of a terminal (including <i>isatty</i> , <i>ttyslot</i> )	UNIX 7	✓
ungetc	push character back into input stream	UNIX 7	✓

#### 4 Special files

cat	phototypesetter interface	UNIX 7	X
cg	ICL 6202 Correspondence Printer on the GPIB	PNX	
cr	ICL 6202 Correspondence Printer on the RS232C	PNX	
dn	DN-11 ACU interface	UNIX 7	X
du	DU-11 201 data phone	UNIX 7	X
err	kernel error reporting	PNX	
flop	PERQ floppy disc	PNX	
gpib	GPIB interface	PNX	

hard	PERQ fixed disc	PNX
hp	RH-11/RP04, RP05, RP06 moving head disc	UNIX 7 X
hs	RH-11/RS03, RS04 fixed head disc	UNIX 7 X
ht	RH-11/TU-16 magtap interface	UNIX 7 X
lp	ICL 3185 Matrix printer	PNX
mem	core memory (including <i>kmem</i> )	UNIX 7 ✓
null	data sink	UNIX 7 ✓
pk	packet driver	UNIX 7 X
rf	RF-11/RS-11 fixed head disc file	UNIX 7 X
rk	RK-11/RK03 or RK05 disc	UNIX 7 X
rp	RP-11/RP03 moving head disc	UNIX 7 X
rs	general purpose RS232C interface driver (including <i>rsa</i> , <i>rsb</i> )	PNX
rstty	general terminal interface via RS232C interface	PNX
screen	screen graphics interface	PNX
speech	general purpose speech driver	PNX
tablet	PERQ tablet	PNX
tc	TC-11/TU56 DEC tape	UNIX 7 X
tm	TM-11/TU-10 magtape interface	UNIX 7 X
ts	transport service interface	PNX
tty	general terminal interface (replaced)	UNIX 7 X
tty	general terminal interface under	PNX *
vp	Versatec printer plotter (replaced)	UNIX 7 X
vp	ICL 6203 Electrostatic Printer/Plotter	PNX *
window	general window interface under	PNX
Z80	special file for loading Z80 firmware	PNX

## 5 File formats and conventions

acct	execution accounting file	UNIX 7 ✓
a.out	assembler and linker output (replaced)	UNIX 7 X
a.out	assembler and linker output	PNX *
ar	archive (library) file format	UNIX 7 ✓
core	format of core image file	UNIX 7 ✓
dir	format of directories	UNIX 7 ✓
dump	incremental dump format (including <i>ddate</i> )	UNIX 7 X
environ	user environment	UNIX 7 ✓
errfile	error log file format	PNX
filsys	format of file system volume (replaced)	UNIX 7 X
filsys	format of file system volume (including <i>fbblk</i> , <i>ino</i> )	PNX *
group	group file	UNIX 7 ✓
mpxio	multiplexed I/O	UNIX 7 X

<b>mtab</b>	mounted file system table	UNIX 7	✓
<b>menufile</b>	input file for <i>buildtext</i>	PNX	
<b>passwd</b>	password file	UNIX 7	✓
<b>plot</b>	graphics interface (replaced)	UNIX 7	X
<b>plot</b>	plot file format	PNX	*
<b>sccsfile</b>	format of sccs file	UNIX 3	
<b>spoolgov</b>	spoolqueue information file	PNX	
<b>tp</b>	DEC/mag tape formats	UNIX 7	X
<b>ttys</b>	terminal initialisation data	UNIX 7	✓
<b>utmp</b>	login records (including <i>wtmp</i> )	UNIX 7	✓
<b>wcurs</b>	PNX WMS cursor patterns	PNX	
<b>wdesc</b>	format of window descriptor file	PNX	
<b>wprofile</b>	PNX WMS initialisation profile	PNX	

**6 Games**

<b>arithmetic</b>		UNIX 7	X
<b>backgammon</b>		UNIX 7	X
<b>banner</b>		UNIX 7	X
<b>bcd</b>		UNIX 7	X
<b>bj</b>		UNIX 7	X
<b>checkers</b>		UNIX 7	X
<b>chess</b>		UNIX 7	X
<b>ching</b>		UNIX 7	X
<b>maze</b>		UNIX 7	X
<b>moo</b>		UNIX 7	X
<b>quiz</b>		UNIX 7	X
<b>reversi</b>		UNIX 7	X
<b>ttt</b>		UNIX 7	X
<b>words</b>		UNIX 7	X
<b>wump</b>		UNIX 7	X

**7 Macro packages and language conventions**

<b>ascii</b>	map of ASCII character set	UNIX 7	✓
<b>eqnchar</b>	special character definitions for <i>eqn</i>	UNIX 7	✓
<b>greek</b>	graphics for extended TTY-37 type box	UNIX 7	X
<b>hier</b>	file system hierarchy	UNIX 7	✓
<b>man</b>	macros to typeset manual	UNIX 7	X
<b>ms</b>	macros for formatting manuscripts	UNIX 7	X
<b>term</b>	conventional names	UNIX 7	X

**8 Maintenance**

<b>boot</b>	startup procedures	UNIX 7	X
-------------	--------------------	--------	---

cron	clock daemon	UNIX 7	✓
crash	what to do when the system crashes	UNIX 7	X
getty	set typewriter mode (replaced)	UNIX 7	X
getty	set typewriter mode	PNX	*
init	process control initialisation (replaced)	UNIX 7	X
init	process control initialisation	PNX	*
makekey	generate encryption key	UNIX 7	X
update	periodically update the superblock	UNIX 7	✓



This appendix contains specifications of items in PNX which are either additional to or are modified versions of the standard UNIX facilities.

These specifications have been produced in the style of Volume 1 of the *UPM* and all start on a new sheet so that you can, if you wish, insert them into your copy of Volume 1 of the *UPM* at or very close to the appropriate place.

Note that store words are 16 bit unless otherwise specified.

#### Contents

ADMIN(1)  
BANNER(1)  
BDIFF(1)  
CC(1)  
CDC(1)  
CEDRA(1)  
CHATTER(1)  
CHROOT(1M)  
COMB(1)  
CPIO(1)  
CPTREE(1)  
CREF(1)  
CSPLIT(1)  
CUT(1)  
DEBUGOFF(1)  
DELTA(1)  
DIFFMK(1)  
DIRCMP(1)  
ENV(1)  
F77(1)  
FL(1)  
FSCK(1)  
GET(1)  
GETOPT(1)  
HELP(1)  
HYPHEN(1)  
ID(1)  
INSTALL(1M)  
LD(1)  
LINE(1)  
LINK(1M)  
LPR(1)  
LSTREE(1)  
MAKEPNX(1M)  
MFTP(1)  
MKFLOP(1)  
MKFS(1M)  
MKNET(1)  
MKWIND(1)  
MOUNTFLOP(1M)  
MVDIR(1M)  
NEWS(1)  
NL(1)  
NM(1)  
PACK(1)  
PASTE(1)  
PLOT(1)  
PRIME(1M)  
PRINTERR(1)  
PRS(1)  
PWCK(1M)  
RANLIB(1)  
REGCMP(1)  
RMDEL(1)  
RSTTY(1)

SACT(1)  
SCCSDIFF(1)  
SDB(1)  
SETPF(1)  
SHUTDOWN(1M)  
SLAVEFT(1)  
SPY(1)  
SRS(1)  
STTY(1)  
UNAME(1)  
UNGET(1)  
VAL(1)  
WHAT(1)  
WHODO(1M)  
WINIT(1)  
WSETUP(1M)  
XARGS(1)  
XREF(1)

IOCTL(2)  
MKNOD(2)  
PTRACE(2)  
SIGNAL(2)  
WDBYTE(2)  
WGREAD(2)  
WLINE(2)  
WRASOP(2)

BUILDTEXT(3)  
CTYPE(3)  
GETOPT(3C)  
LPLOTA(3)  
LPLOTB(3)  
LPQ(3)  
MENUESC(3)  
PICKMENU(3)  
RECTANGLE(3)  
REGEX(3X)  
STRING(3C)

CG(4)  
CR(4)  
ERR(4)  
FLOP(4)  
GPIB(4)  
HARD(4)  
LP(4)  
RS(4)  
RSTTY(4)  
SCREEN(4)  
SPEECH (4)  
TABLET(4)  
TS(4)  
TTY(4)  
VP(4)  
WINDOW(4)  
Z80(4)

A.OUT(5)  
ERRFILE(5)  
FILSYS(5)  
MENUFILE(5)  
PLOT(5)  
SCCSFILE(5)  
SPOOLGOV(5)  
WCURS(5)  
WDESC(5)  
WPROFILE(5)

GETTY(8)  
INIT(8)

**ADMIN(1)****NAME**

admin - create and administer SCCS files

**SYNOPSIS**

```
admin [-n] [-i[name]] [-rrel] [-t[name]] [-fflag[flag-val]]  
[-dflag[flag-val]] [-alogin] [-elogin] [-m[mrlist]]  
[-y[comment]] [-h] [-z] files
```

**DESCRIPTION**

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

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

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

- |                 |   |
|-----------------|---|
| <b>-n</b>       | This keyletter indicates that a new SCCS file is to be created  |
| <b>-i[name]</b> | The <i>name</i> of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see <b>-r</b> keyletter for delta numbering scheme). If the <b>-i</b> letter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an <i>admin</i> command on which the <b>-i</b> keyletter is supplied. Using a single <i>admin</i> to create two or more SCCS files requires that they are created empty (no <b>-i</b> keyletter). Note that the <b>-i</b> keyletter implies the <b>-n</b> keyletter |
| <b>-rrel</b>    | The <i>release</i> into which the initial delta is inserted. This keyletter may be used only if the <b>-i</b> keyletter is also used. If the <b>-r</b> keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1)  |
| <b>-t[name]</b> | The <i>name</i> of a file from which descriptive text for the SCCS file is to be taken. If the <b>-t</b> keyletter is used and <i>admin</i> is creating a new SCCS file (the <b>-n</b> and/or <b>-i</b> keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files:  |
- 1 A **-t** keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file

**ADMIN(1)**

- 2 A -t keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file

**-f**

This keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCS file. Several -f keyletters may be supplied on a single *admin* command line. The allowable *flags* and their values are:

<b>b</b>	Allows use of the -b keyletter on a <i>get(1)</i> command to create branch deltas
<b>cceil</b>	The highest release (that is, "ceiling"), a number less than or equal to 9999, which may be retrieved by a <i>get(1)</i> command for editing. The default value for an unspecified c flag is 9999
<b>ffloor</b>	The lowest release (that is, "floor"), a number greater than 0 but less than 9999, which may be retrieved by a <i>get(1)</i> command for editing. The default value for an unspecified f flag is 1
<b>dSID</b>	The default delta number (SID) to be used by a <i>get(1)</i> command
<b>i</b>	Causes the "No id keywords (ge6)" message issued by <i>get(1)</i> or <i>delta(1)</i> to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see <i>get(1)</i> ) are found in the text retrieved or stored in the SCCS file
<b>j</b>	Allows concurrent <i>get(1)</i> commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file
<b>llist</b>	A list of releases to which deltas can no longer be made ( <i>get -e</i> against one of these "locked" releases fails). The <i>list</i> has the following syntax: $\langle list \rangle ::= \langle range \rangle   \langle list \rangle , \langle range \rangle$ $\langle range \rangle ::= RELEASE NUMBER   a$
	The character <b>a</b> in the <i>list</i> is equivalent to specifying <i>all releases</i> for the named SCCS file
<b>n</b>	Causes <i>delta(1)</i> to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a <i>new</i> release (for example, in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file preventing branch deltas from being created from them in the future
<b>qtext</b>	User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by <i>get(1)</i>
<b>mmod</b>	<i>Module</i> name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by <i>get(1)</i> . If the <b>m</b> flag is not specified, the value assigned is the name of the SCCS file with the leading <b>s.</b> removed
<b>ttype</b>	<i>Type</i> of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by <i>get(1)</i>

**ADMIN(1)**

<b>v[pgm]</b>	Causes <i>delta(1)</i> to prompt for Modification Request ( <i>MR</i> ) numbers as the reason for creating a delta. The optional value specifies the name of an <i>MR</i> number validity checking program (see <i>delta(1)</i> ). (If this flag is set when creating an SCCS file, the <b>m</b> keyletter must also be used even if its value is null)
<b>-dflag</b>	Causes removal (deletion) of the specified <i>flag</i> from an SCCS file. The <b>-d</b> keyletter may be specified only when processing existing SCCS files. Several <b>-d</b> keyletters may be supplied on a single <i>admin</i> command. See the <b>-f</b> keyletter for allowable <i>flag</i> names
<b>l list</b>	A <i>list</i> of releases to be "unlocked". See the <b>-f</b> keyletter for a description of the <b>l</b> flag and the syntax of a <i>list</i>
<b>-a login</b>	A <i>login</i> name, or numerical UNIX group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all <i>login</i> names common to that group ID. Several <b>-a</b> keyletters may be used on a single <i>admin</i> command line. As many <i>logins</i> , or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas.
<b>-e login</b>	A <i>login</i> name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all <i>login</i> names common to that group ID. Several <b>-e</b> keyletters may be used on a single <i>admin</i> command line
<b>-y comment</b>	The <i>comment</i> text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of <i>delta(1)</i> . Omission of the <b>-y</b> keyletter results in a default comment line being inserted in the form:  date and time created YY/MM/DD HH:MM:SS by <i>login</i>
	The <b>-y</b> keyletter is valid only if the <b>-i</b> and/or <b>-n</b> keyletters are specified (that is, a new SCCS file is being created)
<b>-m mrlist</b>	The list of Modification Requests ( <i>MR</i> ) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to <i>delta(1)</i> . The <b>v</b> flag must be set and the <i>MR</i> numbers are validated if the <b>v</b> flag has a value (the name of an <i>MR</i> number validation program). Diagnostics occur if the <b>v</b> flag is not set or <i>MR</i> validation fails
<b>-h</b>	Causes <i>admin</i> to check the structure of the SCCS file (see <i>sccsfile(1)</i> ), and to compare a newly computed checksum (the sum of all the characters in the SCCS file except those in the first line) with the checksum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.
	This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files
<b>-z</b>	The SCCS file checksum is recomputed and stored in the first line of the SCCS file (see <b>-h</b> , above).
	Note that use of this keyletter on a truly corrupted file may prevent future detection of the corruption

## ADMIN(1)

### FILES

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

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

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

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

### SEE ALSO

*delta(1)*, *ed(1)*, *get(1)*, *help(1)*, *prs(1)*, *what(1)*, *sccsfile(5)*.

Bonanni, L.E. and Salemi, C.A. *Source Code Control System User Guide*.

### DIAGNOSTICS

Use *help(1)* for explanations.

**BANNER(1)****NAME**

**banner** - makes posters

**SYNOPSIS**

**banner** [strings]

**DESCRIPTION**

*banner* prints its arguments (each up to 10 characters long) in large letters on the standard output.



**BDIFF(1)****NAME**

**bdiff - big diff**

**SYNOPSIS**

**bdiff file1 file2 [n] [-s]**

**DESCRIPTION**

*bdiff* is used in a manner analogous to *diff(1)* to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for *diff*. *bdiff* ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. The value of *n* is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail. If *file1* (*file2*) is -, the standard input is read. The optional -s (silent) argument specifies that no diagnostics are to be printed by *bdiff* (note, however, that this does not suppress possible exclamations by *diff*). If both optional arguments are specified, they must appear in the order indicated above.

The output of *bdiff* is exactly that of *diff*, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

**FILES**

/tmp/bd??????

**SEE ALSO**

*diff(1)*.

**DIAGNOSTICS**

Use *help(1)* for explanations.



CC(1)

**NAME****cc - C compiler****SYNOPSIS****cc [option] ... file ...****DESCRIPTION**

*cc* is the UNIX C compiler. It accepts several types of arguments:

Arguments whose names end with .c are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with .o substituted for .c. The .o file is normally deleted, however, if a single C program is compiled and loaded all at one go.

The following options are interpreted by *cc* (see *ld*(1) for load-time options):

- c** Suppress the loading phase of the compilation, and force an object file to be produced even if only one program is compiled
- p** Arrange for the compiler to produce code which counts the number of times each routine is called; also, if loading takes place, replace the standard startup routine by one which automatically calls *monitor*(3) 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 use of *prof*(1)
- O** Invoke an object-code optimiser
- P** Run only the macro preprocessor and place the result for each .c file in a corresponding .i file which has no # lines in it
- E** Run only the macro preprocessor and send the result to the standard output. The output is intended for compiler debugging; it is unacceptable as input to *cc*
- o *output*** Name the final output file *output*. If this option is used the file *a.out* will be left undisturbed
- o** This option is for use when only one .c source file is used (for example: *myprog.c*). The final output is placed in a file of the same name, less .c (for example: *myprog*). The file *a.out* is left undisturbed
- D *name=def*** Define the *name* to the preprocessor, as if by #define.  
**-D *name*** If no definition is given, the name is defined as 1
- U *name*** Remove any initial definition of *name*
- I *dir*** #include files whose names do not begin with / are always sought first in the directory of the *file* argument, then in the directories named in -I options, then in directories on a standard list
- B *string*** Find substitute compiler passes in the files named *string* with the suffixes *cpp*, *ccom0*, *ccom1*, and *ccom2*. If *string* is empty, use a standard backup version

## CC(1)

**-t[p012]** Find only the designated compiler passes in the files whose names are constructed by a **-B** option. In the absence of a **-B** option, the *string* is taken to be **/usr/c/**.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier *cc* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **a.out**

## FILES

file.c	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporaries for <i>cc</i>
/lib/cpp	preprocessor
/lib/ccom[01]	compiler for <i>cc</i>
/usr/lib/occom[012]	backup compiler for <i>cc</i>
/usr/lib/ocpp	backup preprocessor
/lib/ccom2	optional optimiser
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/lib/libc.a	standard library, see <i>intro(3)</i>
/usr/include	standard directory for '#include' files

## SEE ALSO

*monitor(3)*, *prof(1)*, *ld(1)*

Kernighan, B.W. and Ritchie, D.M., *The C Programming Language*, Prentice-Hall, 1978

Ritchie, D.M., *C Reference Manual*

## DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader.

**CDC(1)****NAME**

**cdc** - change the delta commentary of an SCCS delta

**SYNOPSIS**

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

**DESCRIPTION**

*cdc* changes the *delta commentary*, for the SID specified by the **-r** keyletter, of each named SCCS file.

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

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

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

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

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

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

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

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

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

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

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

## CDC(1)

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

The exact permissions necessary to modify the SCCS file are documented in the *Source Code System User's Guide*. Simply stated, they are:

- 1 If you made the delta, you can change its delta commentary
- 2 If you own the file and directory you can modify the delta commentary

## EXAMPLES

```
cdc -r1.6 -m"b178-12345 !b177-54321 b179-00001" -ytrouble s.file
```

adds b178-12345 and b179-00001 to the MR list, removes b177-54321 from the MR list, and adds the comment trouble to delta 1.6 of s.file.

```
cdc-r1.6 s.file  
MRs? !b177-54321 b178-12345 b179-00001  
comments? trouble  
does the same thing.
```

## WARNINGS

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

## FILES

x-file (see *delta(1)*)  
z-file (see *delta(1)*)

## SEE ALSO

*admin(1)*, *delta(1)*, *get(1)*, *help(1)*, *prs(1)*, *sccsfile(5)*.  
Bonanni, L.E. and Salemi, C.A., *Source Code Control System User's Guide*

## DIAGNOSTICS

Use *help(1)* for explanations.

**CEDRA(1)****NAME**

*cedra* - cursor editor

**SYNOPSIS**

*cedra*

**DESCRIPTION**

*cedra* is a screen based interactive cursor editor. All editing commands are entered using the puck, either by selecting from a menu or by pressing a particular button in a particular area of the screen. Using *cedra* is described in detail in Chapter 5 of the *Guide to PNX*. *Cedra* commands are described in the help file */usr/lib/cfedra/help*

**FILES**

*/usr/lib/cfedra/help*

**SEE ALSO**

*Guide to PNX*



**CHATTER(1)****NAME**

*chatter* - teletype emulator

**SYNOPSIS**

*chatter*

**DESCRIPTION**

*chatter* provides a completely general teletype emulation using the PERQ keyboard and screen (or window) running to a remote system via the RS232C interface.

Before running *chatter*, the RS232C device options should be set using *srs* to set the appropriate line parameters.

Note: *chatter* cannot cope with continuous input at 9600 baud as the input buffer overflows.

In normal running, except for ^K, the keyboard is completely transparent. Control sequences depend on the remote mainframe.

^K provides a menu of local commands. The commands are:

**Save filename**      Specifies a PNX file to save all mainframe output (transparently) including all echoed input characters.  
                          Prompts for *filename* if not specified

**Transfer filename**      specifies a PNX file to transmit over the RS232C link. This is a transparent transfer, no characters are deleted or substituted. No account is taken of mainframe prompts or other flow control systems.

At end of transmission the message:

**Transmit file closed**

is displayed. The command prompts for the file to transmit  
**Input filename**      This special input mode is designed to allow text input to be sent one line at a time. The program waits for a mainframe prompt string in between each input record. The program prompts for the file to transmit.

After specifying the transmit filename the program prompts:

**Mainframe prompt string:**

The reply should be up to eight characters. After sending a line of text the program monitors output looking for the prompt string. When it is encountered the next line of the input file is sent. The normal prompt from VME2900 Input file command is /-.

Note that the prompt string should not appear in the data being transmitted or echoed characters will be interpreted as a prompt.

*chatter* issues one further prompt:

**Use ETX (^C) (Y/N)?**

## CHATTER(1)

This asks if the ETX character is to be substituted for the LF at the end of the line to be input. The reply should be Y for VME2900. Default is N. TAB characters are replaced with spaces to the next modulo 8 position in the record.

At the end of transmission, the message:

**Transmit file closed**

is output

**Close** Closes the save file if any. If a save file is open, the message  
**Save file closed**

is output. Otherwise the command has no effect

**Quit** This command quits *chatter* and resets the *tty(4)* parameters to the original value.

If *chatter* has been used to connect to a mainframe remember to log out before quitting

All commands can be issued by their initial capital letter.

Any input other than one of the above commands causes the program to re-enter transparent mode.

While running **Transmit** or **Input** the keyboard remains in **-raw** mode so ^C has the normal PNX effect and the connection between the keyboard and the mainframe is suspended.

*chatter* assumes two things about the keyboard interface:

- 1 The normal *tty(4)* setting is **echo**
- 2 The normal mainframe connection is fullduplex/echoplex

To achieve this *chatter* inverts the *tty(4)* **echo** flag for the input device.

*chatter* can connect to a mainframe that does not operate in echoplex mode. Use **sitty** to set **-echo** before running *chatter*.

*chatter* also sets **raw** and **length 0** on entry.

The *tty(4)* parameters present before invoking *chatter* are reloaded when the program terminates normally.

## CHROOT(1M)

### NAME

**chroot** - change root directory for a command

### SYNOPSIS

**chroot newroot command**

### DESCRIPTION

The given command is executed *relative to the new root*. The meaning of any initial slashes (/) in path names is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Notice that:

**chroot newroot command >x**

creates the file x relative to the original root, not the new one.

This command is restricted to the superuser.

The new root path name is always relative to the current root; even if a *chroot* is currently in effect, the *newroot* argument is relative to the current root of the running process.

### SEE ALSO

*chdir(2)*.

### BUGS

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



**COMB(1)****NAME**

**comb** - combine SCCS deltas

**SYNOPSIS**

**Comb [-o] [-s] [-psid] [-clist]** files

**DESCRIPTION**

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

The generated shell procedure is written on the standard output.

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

**-psid**              The SCCS *ID*entification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file

**-clist**              A *list* (see *get(1)* for the syntax of a *list*) of deltas to be preserved. All other deltas are discarded

**-o**                  For each *get-e* generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the **-o** keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file

**-s**                  This argument causes *comb* to generate a shell procedure which, when run, will produce a report giving, for each file; the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

$$100 * (\text{original} - \text{combined}) / \text{original}$$

It is recommended that before any SCCS files are actually combined, one should use this option to determine exactly how much space is saved by the combining process

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

**FILES**

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

**SEE ALSO**

*admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5)*

Bonanni, L.E. and Salemi, C.A., *Source Code Control System User's Guide*

**COMB(1)****DIAGNOSTICS**

Use *help(1)* for explanations.

**BUGS**

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

**CPIO(1)****NAME**

**cpio** - copy file archives in and out

**SYNOPSIS**

**cpio -o [acBv]**  
**cpio -i [Bcdmrtuv6] [patterns]**  
**cpio -p [adlmrvu] directory**

**DESCRIPTION**

**cpio -o** (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information.

**cpio -i** (copy in) extracts from the standard input (which is assumed to be the product of a previous **cpio -o**) the names of files selected by zero or more *patterns* given in the name-generating notation of *sh(1)*. In *patterns*, meta-characters ?, \*, and [...] match the slash / character. The default for *patterns* is \* (that is, select all files).

**cpio -p** (pass) copies out and in, in a single operation. Destination path names are interpreted relative to the named *directory*.

The meanings of the available options are:

- a Reset access times of input files after they have been copied
- B Input/output is to be blocked 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from */dev/rmt?*)
- d *Directories* are to be created as needed
- c Write *header* information in ASCII character form for portability
- r Interactively *rename* files. If the user types a null line, the file is skipped
- t Print a *table of contents* of 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*: causes a list of file names to be printed. When used with the t option, the table of contents looks like the output of an *ls -1* command (see *ls(1)*)
- l Whenever 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 that are being copied
- 6 Process an old (that is, UNIX *Sixth* Edition format) file. Only useful with -i (copy in)

**EXAMPLES**

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls | cpio -o >/dev/mt0  
cd olddir  
find . -print | cpio -pd1 newdr
```

## CPIO(1)

The trivial case

```
find . -print | cpio -oB >/dev/rmt0
```

can be handled more efficiently by:

```
find . -cpio /dev/rmt0
```

## SEE ALSO

*ar(1), find(1), cpio(5).*

## BUGS

Path names are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost. Only the superuser can copy special files.

**CPTREE(1)****NAME**

*cptree* - filestore sub-tree copy

**SYNOPSIS**

*cptree* [flags] input-directory output-directory

**DESCRIPTION**

*cptree* copies the sub-tree specified by input-directory to output-directory. If output-directory does not exist it is created. Sub-directories of output-directory are created as required. The access permissions on the created files are dependent upon the user's *umask* unless the **-t** flag is specified (see below). Special files are not copied.

Warning: Existing copies of files in the output tree are automatically over-written (see **-o** flag).

The optional flags are:

- a**      Include all files whose names start with . (they are otherwise ignored)
- p**      Suppress printing of the commentary
- o**      Do not over-write existing files in the output tree
- u**      Unlink (delete) files in the input tree after they have been successfully copied. Gives the same effect as *mv*
- d**      Include empty directories (they are otherwise ignored)
- D3**     Copy subtree to depth 3.  
The most common use, **-D1**, causes a copy of only the plain files in the input-directory
- M3:7a**   Copy only files last modified after 3 days 7 hours ago
- M3:7b**   Copy only files last modified before 3 days 7 hours ago
- A...**    Same as **-M** flag, with time of last access instead of time of last modification
- t**       Transfer the access permission of the original file to the new file. Also (for superuser only) transfer the ownership to the owner of the original file

**DIAGNOSTICS**

*cptree* gives extensive diagnostics in the event of an error. The error messages are not affected by the **-p** flag.



**CREF(1)****NAME**

*cref* - make cross-reference listing

**SYNOPSIS**

*cref* [ -acilnostux123 ] *files*

**DESCRIPTION**

*cref* makes a cross-reference listing of assembler or C programs; *files* are searched for symbols in the appropriate syntax.

The output report is in four columns:

- 1 symbol
- 2 file name
- 3 text below
- 4 text as it appears in the file

*cref* uses either an *ignore* file or an *only* file. If the *x* option is given, the next argument is taken to be an *ignore* file; if the *-o* option is given, the next argument is taken to be an *only* file. *Ignore* and *only* files are lists of symbols separated by new-lines. All symbols in an *ignore* file are ignored in columns 1 and 3 of the output. If an *only* file is given, only symbols in that file will appear in column 1. Only one of these options may be given; the default setting is *-i* using the default ignore file (see *FILES* below). Assembler pre-defined symbols or C keywords are ignored.

The *-s* option causes current symbols to be put in column 3. In the assembler, the current symbol is the most recent name symbol; in C, the current function name. The *-l* option causes the line number within the file to be put in column 3.

The *-t* option causes the next available argument to be used as the name of the intermediate file (instead of the temporary file */tmp/crt??*). This file is created and is *not* removed at the end of the process.

The *cref* options are:

- a assembler format (default)
- c C format input
- i use an *ignore* file (see above)
- l put line number in column 3 (instead of current symbol)
- n omit column 4 (no context)
- o use an *only* file (see above)
- s current symbol in column 3 (default)
- t user-supplied temporary file
- u print only symbols that occur exactly once
- x print only C external symbols
- 1 sort output on column 1 (default)
- 2 sort output on column 2
- 3 sort output on column 3

**CREF(1)****FILES**

/tmp/crt??	temporaries
/usr/lib/cref/aign	default assembler <i>ignore</i> file
/usr/lib/cref/atab	grammar table for assembler files
/usr/lib/cref/cign	default C <i>ignore</i> file
/usr/lib/cref/ctab	grammar table for C files
/usr/lib/cref/crpost	post-processor
/usr/lib/cref/upost	post-processor for -u option

**SEE ALSO**

*as(1), cc(1), sort(1), xref(1).*

**BUGS**

*cref* inserts an ASCII DEL character into the intermediate file after the eighth character of each name that is eight or more characters long in the source file.

## CSPLIT(1)

### NAME

*csplit* - context split

### SYNOPSIS

*csplit* [-s] [-k] [-f prefix] *file arg1 [...argn]*

### DESCRIPTION

*csplit* reads *file* and separates it into *n+1* sections, defined by the arguments *arg1* ... *argn*. By default the sections are placed in *xx00...xxn* (*n* may not be greater than 99). These sections get the following pieces of *file*:

00: From the start of *file* up to (but not including) the line referenced by *arg1*

01: From the line referenced by *arg1* up to the line referenced by *arg2*

...

*n+1*: From the line referenced by *argn* to the end of *file*

The options to *csplit* are:

- s      *csplit* normally prints the character counts for each file created. If the -s option is present, *csplit* suppresses the printing of all character counts
- k      *csplit* normally removes created files if any error occurs. If the -k option is present, *csplit* leaves previously created files intact
- f *prefix*      If the -f option is used, the created files are named *prefix00...prefixn*. The default is *xx00...xxn*

The arguments (*arg1* ... *argn*) to *csplit* can be a combination of the following:

- /*rexp*/      A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional + or - some number of lines (for example, /Page/-5)
- %*rexp*%      This argument is the same as /*rexp*/, except that no file is created for the section
- /*nno*      A file is to be created from the current line up to (but not including) *nno*. The current line becomes *nno*
- {*num*}      Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *nno*, the file will be split every *nno* lines (*num* times) from that point

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. *csplit* does not affect the original file; it is the users responsibility to remove it.

## CSPLIT(1)

### EXAMPLES

```
csplit -f cobol file '/procedure division/' /par5./ /par16./
```

This example creates four files, **cobol00** ... **cobol03**. After editing the split files, you can recombine them as follows:

```
cat cobol0[0-3]>file
```

Note that this example overwrites the original file.

```
csplit -k file 100 {99}
```

This example would split the file at every 100 lines, up to 10,000 lines. The **-k** option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c '%main(%"/^ }/+1' {20}
```

Assuming that **prog.c** follows the normal C coding convention of ending routines with a **}** at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in **prog.c**.

### SEE ALSO

*ed(1), sh(1), regexp(7).*

### DIAGNOSTICS

Self explanatory except for the message:

**arg - out of range**

which means that the given argument did not reference a line between the current position and the end of the file.

**CUT(1)****NAME**

`cut` - cut out selected fields of each line of a file

**SYNOPSIS**

`cut -clist [file1 file2 ...]`

`cut -flist [-dchar] [-s] [file1 file2 ...]`

**DESCRIPTION**

Use `cut` to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, that is, character positions as on a punched card (-c option), or the length can vary from line to line and be marked with a field delimiter character like *tab* (-f option). `cut` can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

- |                     |   |
|---------------------|---|
| <i>list</i>         | A comma-separated list of integer field numbers (in increasing order), with optional - to indicate ranges as in the -o option of <i>nroff/troff</i> for page ranges; for example, 1,4,7;1-3,8; -5,10(short for 1-5,10); or 3- (short for third through last field)  |
| <code>-clist</code> | The <i>list</i> following -c (no space) specifies character positions (for example, -c1-72 would pass the first 72 characters of each line)   |
| <code>-flist</code> | The list following -f is a list of fields assumed to be separated in the file by a delimiter character (see <code>-d</code> ); for example, -f1,7 copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless <code>-s</code> is specified |
| <code>-dchar</code> | The character following <code>-d</code> is the field delimiter (-f option only). Default is <i>tab</i> . Space or other characters with special meaning to the shell must be quoted   |
| <code>-s</code>     | Suppresses lines with no delimiter characters in case of -f option. Unless specified, lines with no delimiters will be passed through untouched   |

Either the `-c` or `-f` option must be specified.

**HINTS**

Use `grep(1)` to make horizontal cuts (by context) through a file, or `paste(1)` to put files together column-wise (that is, horizontally). To reorder columns in a table, use `cut` and `paste`.

**EXAMPLES**

<code>cut -d: -f1,5 /etc/passwd</code>	mapping of user IDs to names
<code>name = who am i   cut -f1 -d"""</code>	to set <code>name</code> to current login name

**CUT(1)****DIAGNOSTICS**

<b>line too long</b>	A line can have no more than 51 characters or fields
<b>bad list for c/f option</b>	Missing -c or -f option or incorrectly specified <i>list</i> . No error occurs if a line has fewer fields than the <i>list</i> calls for
<b>no fields</b>	The <i>list</i> is empty

**SEE ALSO**

*grep(1), paste(1)*.

**DEBUGOFF(1)****NAME**

**debugoff** - disables the microcode debugger  
**debugon** - enables the microcode debugger

**SYNOPSIS**

**debugoff**  
**debugon**

**DESCRIPTION**

Only the superuser can invoke either *debugon* or *debugoff*. The default state of the microcode debugger is enabled. Pressing ^INS invokes the microcode debugger and pressing G returns you to the system.

Most users do not need the microcode debugger so it is best to disable the debugger using *debugoff*. Holding down ^LF while the debugger is enabled, something which could conceivably happen by accident, would have disastrous results on the system necessitating a reboot.

The debugger can also be enabled or disabled from within a process (see *tty(4)*).

**SEE ALSO**

*tty(4)*



**DELTA(1)****NAME**

*delta* - make a delta (change) to an SCCS file

**SYNOPSIS**

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

**DESCRIPTION**

*Delta* is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get(1)* (called the *g-file*, or generated file).

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

*Delta* may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin(1)*) that may be present in the SCCS file (see -m and -y keyletters below).

Keyletter arguments apply independently to each named file.

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

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

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

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

**-m[mrlist]** If the SCCS file has the v flag set (see *admin(1)*) then a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.

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

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

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

## DELTA(1)

- y[comment]      Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.  
If -y is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.
- p      Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff(1)* format

## WARNINGS

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

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

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

## FILES

All files of the form ?-file are explained in the *Source Code Control System User's Guide*. The naming convention for these files is also described there.

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

## SEE ALSO

*admin(1)*, *bdiff(1)*, *get(1)*, *help(1)*, *prs(1)*, *sccsfile(5)*.

Bonanni, L.E. and Salemi, C.A., *Source Code Control System User's Guide*

## DIAGNOSTICS

Use *help(1)* for explanations.

## DIFFMK(1)

### NAME

*diffmk* - mark differences between files

### SYNOPSIS

*diffmk name1 name2 name3*

### DESCRIPTION

*diffmk* compares two versions of a file and creates a third file that includes "change mark" commands for *nroff(1)* or *troff(1)*. *Name1* and *name2* are the old and new versions of the file. *diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter "change mark" (.mc) requests. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single \*

If you wish, you can use *diffmk* to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

*diffmk old.c new.c tmp; nroff tmp | pr*

where the file **macs** contains:

```
.pl 1  
.ll 77  
.nf  
.eo  
.nc
```

The .ll request might specify a different line length, depending on the nature of the program being printed. The .eo and .nc requests are probably needed only for C programs.

If the characters | and \* are inappropriate, a copy of *diffmk* can be edited to change them (*diffmk* is a shell procedure).

### SEE ALSO

*diff(1)*, *nroff(1)*.

### BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, that is, replacing .sp by .sp 2 will produce a "change mark" on the preceding or following line of output.



**DIRCMP(1)****NAME**

**dircmp** - directory comparison

**SYNOPSIS**

**dircmp dir1 dir2**

**DESCRIPTION**

*dircmp* examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated in addition to a list that indicates whether the files common to both directories have the same contents.

**SEE ALSO**

*cmp(1)*, *diff(1)*.

**ENV(1)****NAME**

**env** - set environment for command execution

**SYNOPSIS**

**env** [-] [name=value] ... [command args]

**DESCRIPTION**

*env* obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The -flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

**SEE ALSO**

*sh(1)*, *exec(2)*, *profile(5)*, *environ(7)*.



**F77(1)****NAME**

f77 - Fortran 77 compiler

**SYNOPSIS**

f77 [option] ... file ...

**DESCRIPTION**

*f77* is the PNX Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with .f are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with .o substituted for .f.

In the same way, arguments whose names end with .c are taken to be C source programs and are compiled, producing an .o file.

The following options have the same meaning as in *cc(1)* (see *ld(1)* for load-time options):

- c              Suppress loading and produce .o files for each source file
- p              Prepare object files for profiling, see *prof(1)*
- O              Invoke an object-code optimiser
- o *output*    Name the final output file *output* instead of **a.out**

The following options are peculiar to *f77*:

- onetrip        Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)
- u              Make the default type of variable UNDEFINED rather than using the default Fortran rules.
- U              To retain uppercase letters
- I2             To make default integer and logical values 16 bits long (short)
- C              Compile code to check that subscripts are within declared array bounds.
- w              Suppress all warning messages. If the option is -w66, only Fortran 66 compatibility warnings are suppressed.
- F              Apply Ratfor preprocessor to relevant files, put the result in the file with the suffix changed to .f, but do not compile
- Rx             Use the string *x* as a Ratfor option in processing .r files.

Other arguments are taken to be either loader option arguments, or F77-compatible object programs, typically produced by an earlier run, or perhaps libraries of F77-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **a.out**.

## F77(1)

### FILES

file.[fr]	input file
file.o	object file
a.out	loaded output
/usr/lib/f77pass1	compiler
/usr/lib/f77pass2	pass 2
/lib/ccom2	optional optimiser
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/lib/libc.a	C library, see section 3

### SEE ALSO

*prof(1), cc(1), ld(1)*

Feldman, S.I. and Weinberger, P.J., *A Portable Fortran 77 Compiler*

### DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

### BUGS

The Fortran 66 subset of the language has been exercised extensively; the newer features have not.

**FL(1)****NAME**

**f1** - floppy format and transfer utility

**SYNOPSIS**

/etc/f1

**DESCRIPTION**

**f1** is a utility to format floppy discs and copy files from the floppy disc to the fixed hard disc or copy files from the fixed disc to the floppy disc. **f1** can also set up an empty RT-11 directory (a POS type directory) on the floppy disc and can list the contents of the floppy disc RT-11 directory on the standard output.

Run the utility by first issuing the **f1** command and then inputting commands, as listed below, from the standard input. Each command is associated with a particular floppy device (see *flop(4)*) and a particular driver routine.

**format [interleave pattern]**

formats the floppy discs with the specified interleave pattern. To format a floppy disc you must first issue the appropriate switches (see below). If you do not specify an interleave pattern, a default operates. Sectors are written in the following order.

1 10 19 2 11 20 3 12 21 4 .....

This default is appropriate for PNX filestore floppies. All new floppy discs must first be formatted

**bformat**

formats the floppy disc with a default interleave suitable for boot floppy discs (single density only). Switches to **bformat** are ignored. Sectors are written in the following order:

1 14 8 21 2 15 9 22 3 .....

Note: The default interleave for **format** does not work for boot floppy discs.

**zero**

creates an empty RT-11 directory. This command destroys the existing contents of the disc

**directory**

lists the contents of a RT-11 floppy disc directory (POS) to the standard output

**get file1 file2**

copies *file1* on the floppy disc to *file2* on the fixed disc

**sget file1 file2**

as above but **sget** strips off the CR character (hex 0D)

**put file1 file2**

copies *file1* from the fixed disc to *file2* on the floppy disc

**verify**

checks the floppy by writing and reading every sector

**check**

reads every block of the disc to check that this is possible

## FL(1)

<b>help</b>	(not the HELP key). Displays information about <i>fl</i>
<b>quit</b>	terminates <i>fl</i>
<b>show</b>	shows the current state of the switches (see below)
<b>density</b>	displays the density of the floppy disc
<b>skew n</b>	skews the format by the number of sectors specified for <i>n</i> (range 1 to 25). A skew of 8 means that, if the first sector on the first track is the first one to be read, the first sector on the second track is the ninth sector.

The default skew is 0

The following 2 switches specify the type of floppy disc that is to be acted on. These switches must be set *before* the format command. The other commands are able to determine the type of disc in use.

<b>/singlesided</b>	accesses a single sided floppy disc
<b>/doublesided</b>	accesses a double sided floppy disc. This is the default so need not be specified
<b>/singledensity</b>	formats a single density floppy disc (with format).
<b>/doubledensity</b>	formats a double density floppy disc (with format). This is the default if /singledensity has not been set

### Notes

- 1 Only floppy discs marked by the manufacturer as double density should be used as double density. Although the PNX software does support double density, the hardware does not support this facility. Use of the facility may result in errors which are in general, recovered by the operating system. However, ICL does not support such use
- 2 You need only supply the minimum number of characters that uniquely identify a command
- 3 For the get, sget and put commands, if file2 is not supplied, the output filename is taken to be the same as file1. This may be a full hierachic name. However, on the floppy, only the name after the final / is used
- 4 The density of the floppy disc is automatically determined by *fl(1)* except for format which requires the appropriate switch to be set

## FILES

<b>/dev/flop</b>	created by /etc/mknod /dev/flop b 1 1
<b>/dev/formatfd</b>	created by /etc/mknod /dev/formatfd b 1 4
<b>/dev/fdplx</b>	created by /etc/mknod /dev/fdplx b 1 5

## SEE ALSO

*mkflop(1)*, *mountflop (1)* and other file system commands if you wish to use a floppy disc as part of the PNX file system after formatting it with *fl*.

**FL(1)****DIAGNOSTICS**

- ?     An invalid command causes a ? to be displayed**
- ??    A non-unique command causes a ?? to be displayed**
- Other diagnostics are self-explanatory.**



## FSCK(1)

### NAME

*fsck* - file system consistency check and interactive repair

### SYNOPSIS

/etc/fsck [-y][-n][-sX][-SX][-t filename][filesystem] ...

### DESCRIPTION

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

*fsck* has more consistency checks than its predecessors *check*, *dcheck*, *fcheck* and *icheck* combined.

The following flags are interpreted by *fsck*:

- y Assume a yes response to all questions asked by *fsck*
- n Assume a no response to all questions asked by *fsck*; do not open the file system for writing
- sX Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the superblock of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system

The **-sX** option allows for creating an optimal free-list organisation. The following form of *X* is supported:

*sblocks-per-cylinder:Blocks-to-skip*

If *X* is not given, the values used when the filesystem was created are used. If these values were not specified, then the value **400:9** is used

- SX Conditionally reconstruct the free list. This option is like **-sX** above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **-S** will force a no response to all questions asked by *fsck*. This option is useful for forcing free list reorganisation on uncontaminated file systems
- t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **-t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **-t** flag, *fsck* will prompt the operator for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

If no filesystems are given to *fsck* then a default list of file systems is read from the file */etc/checklist*.

## FSCK(1)

Inconsistencies checked are as follows:

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

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the *lost+found* directory. The name assigned is the i-node number. The only restriction is that the directory *lost+found* must pre-exist in the root of the filesystem being checked and must have empty slots in which entries can be made. This is accomplished by making *lost+found*, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster.

### FILES

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

### DIAGNOSTICS

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

### BUGS

I-node numbers for . and .. in each directory should be checked for validity.  
-g and -b options from *check* should be available in *fsck*.

**GET(1)****NAME**

**get** - get a version of an SCCS file

**SYNOPSIS**

```
get [-rSID] [-ccutoff] [-ilist] [-xlist] [-aseq-no][-k]
[-e] [-l[p]] [-p] [-m] [-n] [-s] [-b] [-g] [-t] file...
```

**DESCRIPTION**

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

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

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

**-rSID**

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

**-ccutoff**

*Cutoff* date-time, in the form:

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

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

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

**-e**

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

**GET(1)**

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

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

**-b**

Used with the -e keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the b flag is not present in the file (see *admin(1)*) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)

**-ilist**

Note: A branch *delta* may always be created from a non-leaf *delta*  
A *list* of deltas to be included (forced to be applied) in the creation of the generated file. the *list* has the following syntax:

<list> ::= <range> | <list> , <range>  
<range> ::= SID|SID - SID

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1

**-xlist**

A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the -i keyletter for the *list* format

**-k**

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

**-l[p]**

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

**-p**

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

**-s**

Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected

**-m**

Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line

**-n**

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

**-g**

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

**GET(1)**

- t                   Used to access the most recently created ("top") delta in a given release (for example, -r1), or release and level (for example, -r1.2)
- a.seq-no.         The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile(5)*). This keyletter is used by the *comb(1)* command; it is not a generally useful keyletter, and users should not use it. If both the -r and -a keyletters are specified, the -a keyletter is used. Care should be taken when using the -a keyletter in conjunction with the -e keyletter, as the SID of the delta to be created may not be what one expects. The -r keyletter can be used with the -a and -e keyletters to control the naming of the SID of the delta to be created

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

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

**Table 1**  
Determination of SCCS Identification String

<i>SID*</i> <i>Specified</i>	<i>-b Keyletter Used†</i>	<i>Other Conditions</i>	<i>SID Retrieved</i>	<i>SID of Delta to be created</i>
none‡	no	R defaults to mR	mR.mL	mR.(mL + 1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB + 1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL + 1)
R	yes	R > mR	mR.mL	mR.mL.(mB + 1).1
R	yes	R = mR	mR.mL	mR.mL.(mB + 1).1
R	-	R < mR and R does not exist	hR.mL**	hR.mL.(mB + 1).1
R	-	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB + 1).1
R.L	no	No trunk succ.	R.L	R.(L + 1)
R.L	yes	No trunk succ.	R.L	R.L(mB + 1).1
R.L	-	Trunk succ. in release > R	R.L	R.L.(mB + 1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS + 1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB + 1).1
R.L.B.S.	no	NO branch succ.	R.L.B.S	R.L.B.(S + 1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB + 1).1
R.L.B.S	-	Branch succ.	R.L.B.S	R.L.(mB + 1).1

**GET(1)**

- \* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L(mB + 1).1" means "the first sequence number on the *new* branch (for example maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or R.L.B.S", each of the specified components *must* exist
- \*\* "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R
- \*\*\* This is used to force creation of the *first* delta in a *new* release
- # Successor
- † The -b keyletter is effective only if the b flag (see *admin(1)*) is present in the file. An entry of - means "irrelevant"
- ‡ This case applies if the d (default SID) flag is *not* present in the file. If the d flag is present in the file, the SID obtained from the d flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies

**IDENTIFICATION KEYWORDS**

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

<i>Keyword</i>	<i>Value</i>
%M%	Module name: either the value of the m flag in the file (see <i>admin(1)</i> ), or if absent, the name of the SCCS file with the leading s. removed
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text
%R%	Release
%L%	Level
%B%	Branch
%S%	Sequence
%D%	Current date (YY/MM/DD)
%H%	Current date (MM/DD/YY)
%T%	Current time (HH:MM:SS)
%E%	Date newest applied delta was created (YY/MM/DD)
%G%	Date newest applied delta was created (MM/DD/YY)
%U%	Time newest applied delta was created (HH:MM:SS)
%Y%	Module type: value of the t flag in the SCCS file (see <i>admin(1)</i> )
%F%	SCCS file name
%P%	Fully qualified SCCS filename
%Q%	The value of the q flag in the file (see <i>admin(1)</i> )

**GET(1)**

<b>%C%</b>	Current line number. This keyword is intended for identifying messages output by the program such as "this shouldn't have happened" type errors. It is not intended to be used on every line to provide sequence numbers
<b>%Z%</b>	The 4-character string @(#) recognisable by <i>what(1)</i>
<b>%W%</b>	A shorthand notation for constructing <i>what(1)</i> strings for UNIX program files. %W% = %Z%%M%<horizontal-tab>%I%
<b>%A%</b>	Another shorthand notation for constructing <i>what(1)</i> strings for non-UNIX program files. %A% = %Z%%Y%%M%%I%%Z%

**FILES**

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

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

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

Lines in the l-file have the following format:

- a    A blank character if the data was applied;  
\* otherwise
- b    A blank character if the delta was applied or wasn't applied and ignored;  
\* if the delta wasn't applied and wasn't ignored
- c    A code indicating a "special" reason why the delta was or was not applied  
"I":Included  
"X":Excluded  
"C":Cut off (by a -c keyletter)
- d    Blank
- e    SCCS identification (SID)
- f    Tab character
- g    Date and time (in the form YY/MM/DD HH:MM:SS) of creation
- h    Blank
- i    Login name of person who created *delta*

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

## GET(1)

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

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

## SEE ALSO

*admin(1), delta(1), help(1), prs(1), what(1), sccsfile(5).*

Bonanni, L.E. and Salemi, C.A., *Source Code Control System User's Guide*

## DIAGNOSTICS

Use *help(1)* for explanations.

## BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, then only one file may be named when the -e keyletter is used.

**GETOPT(1)****NAME**

*getopt* - parse command options

**SYNOPSIS**

```
set - `getopt optstring $*`
```

**DESCRIPTION**

*getopt* is used to break up options in command lines for easy parsing by shell procedures, and to check for legal options. *optstring* is a string of recognised option letters (see *getopt(3C)*); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option -- is used to delimit the end of the options. *getopt* will place -- in the arguments at the end of the options, or recognise it if used explicitly. The shell arguments (\$1,\$2...) are reset so that each option is preceded by a - and in its own shell argument; each option argument is also in its own shell argument.

**EXAMPLE**

The following code fragment shows how one might process the arguments for a command that can take the options a and b, and the option o, which requires an argument.

```
set --getopt abo: $*
if [ $?!= 0 ]
then
    echo $USAGE
    exit 2
fi
for i in $*
do
    case $i in
        -a | -b)
            FLAG=$i; shift;;
        -o)
            OARG=$2; shift; shift;;
        -)
            shift;break;;
    esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg - file file
```

**SEE ALSO**

*sh(1)*, *getopt(3C)*.

**DIAGNOSTICS**

*getopt* prints an error message on the standard error when it encounters an option letter not included in *optstring*.



**HELP(1)****NAME**

**help - ask for help**

**SYNOPSIS**

**help [args]**

**DESCRIPTION**

**help** finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, **help** will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1       Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (for example, **ge6**, for message 6 from the **get** command)

type 2       Does not contain numerics (as a command, such as **get**)

type 3       Is all numerics (for example, **212**)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try

**help stuck**

**FILES**

/usr/lib/help directory contains files of message text.

**DIAGNOSTICS**

Use **help(1)** for explanations.



**HYPHEN(1)****NAME**

**hyphen** - find hyphenated words

**SYNOPSIS**

**hyphen** files

**DESCRIPTION**

*hyphen* finds all the hyphenated words in *files* and prints them on the standard output. If no arguments are given, the standard input is used. Thus *hyphen* may be used as a filter.

**BUGS**

*Hyphen* cannot cope with hyphenated *italic* words; it will often miss them completely, or mangle them.

*Hyphen* occasionally gets confused, but with no ill effects other than spurious extra output.



**ID(1)****NAME**

**id** - print user and group IDs and names

**SYNOPSIS**

**id**

**DESCRIPTION**

*id* writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

**SEE ALSO**

*getuid(2), getgid(2).*



**INSTALL(1M)****NAME**

install - install commands

**SYNOPSIS**

```
install [-c dira] [-f dirb] [-i] [-n dirc] [-o] [-s]
file [dirx...]
```

**DESCRIPTION**

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

If no options or directories (*dirx...*) are given, *install* will search (using *find(1)*) a set of default directories (*/bin*, */usr/bin*, */etc*, */lib*, and */usr/lib*, in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

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

The meanings of the options are:

- |               |   |
|---------------|---|
| <b>-cdira</b> | Installs a new command in the directory specified in <i>dira</i> . Looks for <i>file</i> in <i>dira</i> and installs it there if it is not found. If it is found, <i>install</i> issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the <b>-s</b> option   |
| <b>-fdirb</b> | Forces <i>file</i> to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to 755 and bin, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the <b>-o</b> or <b>-s</b> options |
| <b>-i</b>     | Ignores default directory list, searching only through the given directories ( <i>dirx...</i> ). May be used alone or with any other options other than <b>-c</b> and <b>-f</b>   |
| <b>-ndirc</b> | If <i>file</i> is not found in any of the searched directories, it is put in the directory specified in <i>dirc</i> . The mode and owner of the new file will be set to 755 and bin, respectively. May be used alone or with any other options other than <b>-c</b> and <b>-f</b>   |
| <b>-o</b>     | If <i>file</i> is found, this option saves the "found" file by copying it to OLD <i>file</i> in the directory in which it was found. May be used alone or with any other options other than <b>-c</b>   |
| <b>-s</b>     | Suppresses printing of messages other than error messages. May be used alone or with any other options  |

**SEE ALSO**

*mk(8)*.



**LD(1)****NAME**

*ld* - loader

**SYNOPSIS**

*ld* [option] file ...

**DESCRIPTION**

*ld* combines several object programs into one, resolves external references, and searches libraries. In the simplest case, several object *files* are given, and *ld* combines them, producing an object module which can be executed. The output of *ld* is left on *a.out*. This file is made executable only if no errors occurred during the load.

Object modules and libraries are processed in the order specified, to create the executable file. *ld* checks whether the largest procedure is smaller than the default page size of 8Kb, and if not adjusts the page size to the next multiple of 512 bytes in which the largest procedure can fit. Note that when packing procedures into pages, no optimisation is done; if the next procedure will not fit in the current page, a new page is started.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by *ranlib(1)*, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. If the member of a library is named '*\_\_.SYMDEF*', then it is understood to be a dictionary for the library such as produced by *ranlib*; the dictionary is searched iteratively to satisfy as many references as possible.

The symbols '*\_\_etext*', '*\_\_edata*' and '*\_\_end*' ('*etext*', '*edata*' and '*end*' in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols. The reserved symbol '*\_\_errno*' is associated with a fixed location (known to the kernel and microcode) and no source program should define it.

The loader also assists in the paging of text by the kernel, ensuring that text procedures do not cross a page boundary. When such a procedure is encountered, the text file is padded out to the next page boundary by inserting NULL instructions.

The standard options *-i* and *-n* are not relevant to PNX since all executable files have separate text and data segments (the data starting at location 0 as for *-n*) and so these options are ignored. The standard options *-d*, *-D* and *-O* are not supported.

*ld* understands several options. Except for *-l*, they should appear before the file names.

- s     'Strip' the output, that is remove the symbol table and relocation bits to save space (but impairs the usefulness of the debugger). This information can also be removed by *strip(1)*
- u     Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine

**LD(1)**

- lx This option is an abbreviation for the library name `lib/libx.a', where x is a string. If that does not exist, *ld* tries `/usr/lib/libx.a'. A library is searched when its name is encountered, so the placement of a -l is significant
- x Do not preserve local (non-global) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file
- X Save local symbols except for those whose names begin with L. This option is the default. Names beginning with L *cannot* be saved
- o The *name* argument after -o is used as the name of the *ld* output file, instead of **a.out**
- e The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default
- k Suppresses text paging allowing the kernel to be up to 128Kb. This flag must be specified when linking a kernel. It must *not* be specified when creating any other loadable binary file

**FILES**

/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
a.out	output file

**SEE ALSO**

*as(1), ar(1), cc(1), ranlib(1).*

**LINE(1)****NAME**

line - read one line

**SYNOPSIS**

line

**DESCRIPTION**

*Line* copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

**SEE ALSO**

*sh(1), read(2).*



**LINK(1M)****NAME**

*link, unlink - exercise link and unlink system calls*

**SYNOPSIS**

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

**DESCRIPTION**

*link* and *unlink* perform their respective system calls on their arguments, abandoning all error checking. These commands may only be executed by the superuser, who (it is hoped) knows what he or she is doing.

**SEE ALSO**

*rm(1), link(2), unlink(2).*



**LPR(1)****NAME**

lpr - line printer spooler  
lpc - correspondence printer spooler  
lpp - plotter spooler

**SYNOPSIS**

**lpr** [option]....[file]....  
**lpc** [option]....[file]....  
**lpp** [option]....[file]....

**DESCRIPTION**

The spoolers cause the *files* to be queued for printing or plotting. If no files are named, the standard input is read. The following options are available:

- r Remove the file when it has been queued
- c Copy the file to insulate against changes that may happen before printing
- m Report by *mail(1)* when printing is complete
- n Do not report by *mail*. This is the default option

The file */etc/spoolgov* is read to determine which despooler program is to print or plot files from the spool queue associated with each spooler. If the spool queue is marked as Active in */etc/spoolgov* then the despooler program will be invoked by *exec1* (see *exec(2)*) when a file has been added to the spool queue. If the queue is marked as Closed then the spooler will not add files to its spool queue. Files are queued using *lpq(3)*.

**FILES**

<i>/etc/spoolgov</i>	- printer and plotter spool control file
<i>/usr/lib/lpdml</i>	- 3185 matrix printer despooler
<i>/usr/lib/lpdel</i>	- 6203 electrostatic print despooler
<i>/usr/spool/lpd/lock</i>	
<i>/usr/spool/lpd/cf*</i>	- print data file (copied)
<i>/usr/spool/lpd/lp*</i>	- print data file (linked)
<i>/usr/spool/lpd/df*</i>	- print daemon control file
<i>/usr/spool/lpd/tf*</i>	- print daemon control file, temporary version
<i>/usr/lib/lpdcr</i>	- 6202/03 correspondence print despooler
<i>/usr/lib/lpdcg</i>	- 6202/02 correspondence print despooler
<i>/usr/spool/lpdc/lock</i>	
<i>/usr/spool/lpdc/cf*</i>	- correspondence print data file (copied)
<i>/usr/spool/lpdc/lf*</i>	- correspondence print data file (linked)
<i>/usr/spool/lpdc/df*</i>	- correspondence print daemon control file
<i>/usr/spool/lpdctf*</i>	- correspondence print daemon control file, temporary version
<i>/usr/lib/lpdmp</i>	- 3185 matrix plot despooler
<i>/usr/lib/lpdep</i>	- 6203 electrostatic plot despooler
<i>/usr/spool/lpdः/lock</i>	
<i>/usr/spool/lpdः/cf*</i>	- plot data file (copied)
<i>/usr/spool/lpdः/lf*</i>	- plot data file (linked)
<i>/usr/spool/lpdः/df*</i>	- plot daemon control file
<i>/usr/spool/lpdः/tf*</i>	- plot daemon control file, temporary version



**LSTREE(1)****NAME**

*lstree* - list the contents of any subtree of the PNX file system

**SYNOPSIS**

*lstree directory*

**DESCRIPTION**

*lstree* lists the contents of *directory*, followed by the contents of each subdirectory and so on until all the files known to the directories stemming from the node *directory* have been listed.

*lstree* is not particularly useful for listing the contents of nodes high in the PNX directory hierarchy as it is very slow for large listings.



**MAKEPNX(1M)****NAME**

**makepxn** - establish a PNX system on the hard disc.  
**updatepxn** - update the system leaving the disc intact.

**SYNOPSIS**

**makepxn**  
**updatepxn**

**DESCRIPTION**

*makepxn* initialises the fixed disc, sets up a bootstrap file and copies the filestore from the floppy disc. *makepxn* destroys the previous contents of the fixed disc. It must only be used when the system has been bootstrapped from floppy disc.

*updatepxn* should be used instead of *makepxn* if you are moving from an older PNX as it leaves the fixed disc intact and only overwrites the system files.



**MFTP(1)****NAME**

**mftp** - call another PNX system on an Open Systems LAN

**SYNOPSIS**

**mftp [filename]**

**DESCRIPTION**

*mftp* calls up another PNX system on another PERQ on the local area network to initiate a file transfer. The machine issuing the *mftp* call is the Master for the connection. *mftp* uses the *slaveft* facility in the remote machine to create an interactive PERQ to PERQ conversation. (See under *slaveft(1)* in this guide). *mftp* uses the ECMA 72 Class 4 LAN transport service provided on the PERQ to transmit files from one PNX PERQ to another PNX PERQ. The transport service transmits the files one frame at a time and will wait for an acknowledgment from the remote machine before transmitting the next frame. This ensures that no frames are lost during transmission. If transmission fails, *mftp* is abandoned with the appropriate error message (see list of error codes under *DIAGNOSTICS* at the end of this specification). For further information in the transport service see the publication *ICL PERQ: Establishing IPA under PNX*.

*mftp* may be used as part of a shellscript program if required.

*filename* designates another PERQ on the LAN. The filenames of all the PERQs on the local network are set up together with their system addresses using *mknet(1)*.

If the filename does not exist the error message:

**cannot make connection due to invalid name**

appears. If the filename is omitted the error message:

**destination address omitted**

appears.

Other error messages that may be displayed if the connection is unsuccessful are:

**cannot make connection due to no ethernet board**

**cannot make connection due to file already open**

**cannot make connection due to no response from remote end**

**cannot make connection due to no room in transport layer**

**cannot make connection due to unspecified reason**

and the *mftp* program terminates with exit code 11. (See under *DIAGNOSTICS* at the end of this specification).

The following dialogue reads commands from the standard input file and writes the responses to the standard output file. The standard error file is not used. When using *mftp* the user always has to login to the remote machine first.

If the connection is successful the following prompt appears:

**specify login name for remote PERQ**

## **MFTP(1)**

The user must specify the username to be used in the remote machine. For example, /ftfred. If the username is valid, that is, it is not already in use, and it conforms to the allowed convention the following message appears:

**remote pathname is /**

indicating a successful login to the remote PERQ.

If the username specified is invalid, one of the following error messages appears:

**remote set uid fails  
protocol error**

**Retry the login procedure again when the login prompt reappears.**

**On a successful login the prompt:**

**password :**

appears. The user must specify the password for the user requested in the remote machine. The password is not echoed to the screen. If the login sequence fails due to an invalid password, the request for the login name is repeated. If on retrying the password used is again invalid, one of the following error messages will appear:

**remote set uid fails  
protocol error**

and *mftp* is abandoned. If it is correct, the current remote pathname is displayed and any of the following commands may be typed in response to a > prompt:

**q** Quits MFTP

**crd *directoryname*** Changes the remote directory name. If the directory does not exist, it is not created. If the *crd* call fails, the following message is displayed:

remote chdir call fails

**cld** *directoryname*

**Changes the local directory name. If the local call fails, the following message is displayed:**

**cld** *directoryname*

31 32

**Transfers the sourcefile to the remote machine. If**

the *destfile* is omitted.

*put srcefile desyfile*

Transfers the *sourcfile* to the remote machine. If the *destfile* is omitted, *srcfile* is used in the remote machine. On completion the message:

**copying complete**

is displayed.

The remote file is created with the same access rights as it has locally and the owner will be the name of the remote login name.

If the file in a PNX destination machine already exists, it is overwritten. Should the remote file fail to open, that is, if write permission is not given to the logged in username, the following message is displayed:

**unable to create file in remote machine**

**MFTP(1)**

**get** *srcefile destfile*

Transfers the sourcefile from the remote machine to the local machine. If the *destfile* is omitted, *srcefile* is used in the local machine.

If the file does not exist in the remote machine the following message is displayed:

**file does not exist in remote machine**

If the local file already exists it is overwritten and the message:

**copying complete**

is displayed

**app** *srcefile destfile*

Appends the source file to the end of the destination file in the remote machine. If the destination file does not exist the effect is the same as that of the **put** command.

If the *destfile* parameter is omitted, it is assumed to be the same as the *sourcefile* parameter

Transfers the local file to the remote machine and adds it to the print queue (see below Printing and Plotting)

**print** *localfilename*

Transfers the local file to the remote machine and adds it to the plot queue (see below Printing and Plotting)

*Printing and Plotting*

After a print or plot command *mftp* opens the file */lan/myname* in the remote machine to the printer or plotter, and reads the single line which identifies the caller. This is used for returning printout queued by the print or plot commands.

Should the file */lan/myname* not exist a pair of question marks will be transmitted to the printer instead of the contents of file */lan/myname*. This is a way of telling the user that the program cannot specify which machine the file comes from.

This process is used to print *banner(1)* headlines (see also under printer spooler in this Guide)

## MFTP()

### FILES

/lan/myname,  
/etc/slaveft  
/user/spool

### SEE ALSO

slaveft(1)

### DIAGNOSTICS

If *mftp* runs normally nothing is returned. If *mftp* is abandoned one of the following exit codes may be returned. The exit code is sent to the standard output file. If *mftp* is run from a shellscript the exit code is returned to the shellscript program. The user must correct the error if possible and then run *mftp* again.

<i>Exit Code</i>	<i>Description</i>
0	no errors occurred
1	connection broken due to network failure
2	connection broken due to protocol error
3	unable to make connection to requested PERQ
4	at some stage a local file failed to open for input
5	at some stage a remote file failed to be created
6	connection broken due to remote machine rejecting block
7	at some stage a remote file failed to open for input
8	at some stage a local file failed to be created
9	at some stage an invalid password was supplied
10	connection broken due to local machine rejecting block
11	destination machine omitted from command line argument
12	at some stage a remote setuid failure occurred
13	at some stage a local change directory failed
14	at some stage a remote change directory failed

### BUGS

Remote setuid fails has been known to occur if a copy of *slaveft* is exec'd (see under *exec(1)*, from a user other than the superuser. This can only occur if the copy originally exec'd by the superuser failed to offer a connection.

The present cure is to kill *slaveft* in the remote machine and reexec it from the superuser.

**MKFLOP(1)****NAME**

*mkflop* - creates an empty PNX filestore floppy

**SYNOPSIS**

**mkflop [-f][-o][-s][-rtll][-b]**

**DESCRIPTION**

*mkflop* may be used to fomat and set up a file store floppy disc for use with PNX. If all options are omitted, *mkflop* formats and sets up a double density PNX style file store floppy.

- f      Switches off the format operation (used if disc already formatted using *fl(1)* **format** command). File store is created as specified by other options
- o      Formats disc in old style PNX form, that is, with software interleaving and single density only
- s      Creates a single density floppy. File store type is determined by default or by **-rtll** option
- rtll     Sets up an RT-II (POS) type filestore
- b      Sets up a boot floppy (single density only)

**SEE ALSO**

*flop(4)*, *mountflop(1)*, *fl(1)*.



**MKFS(1M)****NAME**

*mkfs* - construct a file system

**SYNOPSIS**

/etc/mkfs special proto

**DESCRIPTION**

*mkfs* constructs a file system by writing on the special file *special* according to the directions found in the prototype *proto*. The prototype file contains tokens separated by spaces or new lines. The first token is the name of a file to be copied onto block zero as the bootstrap program. The second token is a number specifying the size of the created file system. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of i-nodes in the i-list. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user-id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters -bcd specify regular, block special, character special and directory files respectively.) The second character of the type is either u or - to specify set-id mode or not. The third is g or - for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions, see *chmod(1)*.

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, *mkfs* makes the entries for . and .. and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token \$.

If the prototype file cannot be opened and its name consists of a string of digits, *mkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *proto* interpreted as a decimal number. The number of i-nodes is calculated as a function of the filesystem size. The boot program is left uninitialized.

A sample prototype specification follows:

```
/usr/mdec/uboot
4872 55
d-777 3 1
usr d-777 3 1
    sh  --755 3 1 /bin/sh
    ken d-755 6 1
    $
    b0  b--644 3 1 0 0
    c0  c-644 3 1 0 0
    $
$
```

**SEE ALSO**

*filsys(5),dir(5)*



**MKNET****NAME**

**mknet** - set up transport address file

**SYNOPSIS**

**mknet**

**DESCRIPTION**

The *mknet* program is used to set up the transport address file in each PERQ on the network. The transport address file contains the address that the network controller uses to write data to the desired machine. The transport address file also holds details of whether a PERQ is to be used as a master or a slave machine. For details of how to run *mknet* and the operator responses see the publication *ICL PERQ: Establishing IPA under PNX*.

**SEE ALSO**

*mftp(1)*, *slaveft(1)*, PERQFTP in the publication *ICL PERQ: System Software Reference (POS)* (Edition 4, R10101/00).



**MKWIND(1)****NAME**

**mkwind** - make a window descriptor special file

**SYNOPSIS**

**mkwind** *sfile*

**DESCRIPTION**

*mkwind* creates a window descriptor special file *sfile* from parameters specified in the standard input.

The window properties to define a window are listed as a set of keywords in Table A3.1. Each keyword takes parameters to define the value of that particular property but you do not have to specify a value as there are defaults. If the standard input is the keyboard, issue the *mkwind* command and specify a file to contain the description. Then type input lines of the format:

*keyword* [*param*]

When you have finished entering keywords and their values, type *^Z* to show the end of the input. If you type anything other than a keyword input line or *^Z*, *mkwind* rejects the input and gives you an error message on the standard error file.

The keyword *,help*, displays a list of keywords together with their parameter types on the standard error file.

If the standard input is a file, use the PNX editor to set up the input file. The format should be one or more text lines with the format described above.

The following table shows valid keywords, the parameter type expected, default values generated in the window descriptor if that keyword is not present, and the meaning of those defaults.

**Table A3.1**  
Window properties

keyword	type	default	meaning of default
xstart	integer	0	initial left x co-ordinate
ystart	integer	0	initial top y co-ordinate
width	integer	750	width in pixels
height	integer	500	height in pixels
rank	integer	0	foremost in display
cumode	integer	0	CMAUTO
cufunc	integer	4	CFNORMAL
border	y/n	y	bordered
unique	y/n	n	generic
delete	y/n	n	preserve when closed
title	string	null	untitled
font	file	null	standard font
cupat	file	null	standard selected cursor

Keywords may be in any order. The file *sfile* is defined in *wdesc(5)*.

Values for *cumode*:

CMAUTO	0
CMXAUTO	1
CMON	2
CMOFF	3

## MKWIND(1)

Values for cufunc:

CFBLACKHOLE	2	(Inverted screen and normal cursor, logical AND)
CFWHITEHOLE	3	(Normal screen and inverted cursor, logical AND)
CFNORMAL	4	(Normal screen, cursor superimposed)
CFINVERT	5	(Inverts screen, cursor superimposed)
CFCURSCOMP	6	(Normal screen, cursor opposite to background)
CFINVCURCOMP	7	(Inverted screen, cursor opposite to background)

For font and cupat, the text parameter is a filename. The file should contain the required font or cursor pattern.

## FILES

/bin/mkwind

## SEE ALSO

*window(4), wdesc(5).*

**MOUNTFLOP(1)****NAME**

*mountflop*, *umountflop* - mount and dismount a floppy disc

**SYNOPSIS**

**mountflop[-b][-o][-r]**  
**umountflop**

**DESCRIPTION**

*mountflop* mounts a PNX file store floppy disc. Files on the floppy disc can now be accessed as files in the directory */fd*. *mountflop* is equivalent to:

*mount /dev/floppytype /fd*

If there is no floppy in the drive, *mountflop* gives an error message. Options to *mountflop* specify the type of floppy disc to be mounted and, therefore, the device *mountflop* accesses (see *flop(4)*).

- b specifies a boot floppy
- o specifies old-style PNX file store floppy (pre-release 2.0)
- r specifies floppy is to be mounted for read access only

The default is no options, that is, a PNX style file store floppy, single or double density. *umountflop* unmounts a floppy disc and must be called before removing the floppy from the drive.

Any files in */fd* are left unaffected, but they are not available while the floppy disc is mounted.

**FILES**

*/fd* the directory to receive the mounted file system

**SEE ALSO**

*floppy(4)*, *mkflop(1)*

**DIAGNOSTICS**

Error messages are produced (as with *mount*) if the file system is already mounted, or if the file system is busy while attempting to unmount it.

**BUGS**

Giving the wrong options to *mountflop* is not detected and may cause corruption of the floppy disc. For safety use the -r option.

Removing the floppy from the drive without unmounting it can cause it to be corrupted.



**MVDIR(1M)****NAME**

**mvdir** - move a directory

**SYNOPSIS**

**/etc/mvdir dirname name**

**DESCRIPTION**

*mvdir* renames directories within a file system. *dirname* must be a directory; *name* must not exist. Neither name may be a sub-set of the other (/x/y cannot be moved to /x/y/z, nor vice versa).

Only superuser can use *mvdir*.

**SEE ALSO**

*mkdir(1)*.



**NEWS(1)****NAME**

news - print news items

**SYNOPSIS**

news [-a] [-n] [-s] [items]

**DESCRIPTION**

*news* is used to keep the user informed of current events. By convention, these events are described by files in the directory */usr/news*.

When invoked without arguments, *news* prints the contents of all current files in */usr/news*, most recent first, with each preceded by an appropriate header. *news* stores the "currency" time as the modification date of a file named *.news\_time* in the user's home directory (the identity of this directory is determined by the environment variable **\$HOME**); only files more recent than this currency time are considered "current."

The **-a** option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

The **-n** option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

The **-s** option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's *.profile* file, or in the system's */etc/profile*.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

**FILES**

*/etc/profile*  
*/usr/news/\**  
*\$HOME/.news\_time*

**SEE ALSO**

*profile(5)*, *environ(7)*.



**NL(1)****NAME**

**nl** - line numbering filter

**SYNOPSIS**

```
nl [-htype] [-btype] [-ftype] [-vstart#] [-iincr] [-p]
[-lnum] [-ssep] [-wwidth] [-nformat] file
```

**DESCRIPTION**

*nl* reads lines from the named *file*, or the standard input if no *file* is named, and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

*nl* views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (for example, no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signalled by input lines containing nothing but the following character(s):

<i>Line contents</i>	<i>Start of</i>
\:\:\:	header
\:\:	body
\:	footer

Unless signalled otherwise, *nl* assumes the text being read is in a single logical page body. Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

<b>-btype</b>	Specifies which logical page body lines are to be numbered. Recognised types and their meaning are: <b>a</b> , number all lines; <b>t</b> , number lines with printable text only; <b>n</b> , no line numbering; <b>pstring</b> , number only lines that contain the regular expression specified in <i>string</i> . Default type for logical page body is <b>t</b> (text lines numbered)
<b>-htype</b>	Same as <b>-btype</b> except for header. Default type for logical page header is <b>n</b> (no lines numbered)
<b>-ftype</b>	Same as <b>-btype</b> except for footer. Default for logical page footer is <b>n</b> (no lines numbered)
<b>-p</b>	Do not restart numbering at logical page delimiters
<b>-vstart#</b>	<i>Start#</i> is the initial value used to number logical page lines. Default is 1
<b>-iincr</b>	<i>Incr</i> is the increment value used to number logical page lines. Default is 1
<b>-ssep</b>	<i>Sep</i> is the character(s) used in separating the line number and the corresponding text line. Default <i>sep</i> is a tab
<b>-wwidth</b>	<i>Width</i> is the number of characters to be used for the line number. Default <i>width</i> is 6

**NL(1)**

- nformat      *format* is the line numbering format. Recognised values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes suppressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified)
- lnum      *num* is the number of blank lines to be considered as one. For example, -12 results in only the second adjacent blank being numbered (if the appropriate -ha, -ba, and/or -fa option is set). Default is 1

**SEE ALSO**

*pr(1)*.

**NM(1)****NAME**

**nm** - print name list

**SYNOPSIS**

**nm [-gnopru] [file...]**

**DESCRIPTION**

*Nm* prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in *a.out* are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters U (undefined), A (absolute), T (text segment symbol), D (data segment symbol), B (bss segment symbol), or C (common symbol). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

- g Print only global (external) symbols
- n Sort numerically rather than alphabetically
- o Prepend file or archive element name to each output line rather than only once
- p Don't sort; print in symbol-table order
- r Sort in reverse order
- u Print only undefined symbols

Note: Version numbers of compilers and assemblers only appear if you use the -p option.

**SEE ALSO**

*ar*(1), *ar*(5), *a.out*(5)



**PACK(1)****NAME**

**pack, pcat, unpack - compress and expand files**

**SYNOPSIS**

```
pack [-] name...
pcat name...
unpack name...
```

**DESCRIPTION**

*pack* attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name.z* with the same access modes, access and modified dates, and owner as those of *name*. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

*Pack* uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the - argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of - in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each .z file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

*Pack* returns a value that is the number of files that it failed to compress.

No packing will occur if:

- the file appears to be already packed
- the file name has more than 12 characters
- the file has links
- the file is a directory
- the file cannot be opened
- no disk storage blocks will be saved by packing
- a file called *name.z* already exists
- the .z file cannot be created
- an I/O error occurred during processing

The last segment of the file name must contain no more than 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

## PACK(1)

*pcat* does for packed files what *cat(1)* does for ordinary files. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name.z* use:

**pcat name.z**

or just:

**pcat name**

To make an unpacked copy, say *nnn*, of a packed file named *name.z* (without destroying *name.z*) use the command:

**pcat name>nnn**

*pcat* returns the number of files it was unable to unpack. Failure may occur if:

the file name (exclusive of the .z) has more than 12 characters

the file cannot be opened

the file does not appear to be the output of *pack*

*Unpack* expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name.z* (or just *name*, if *name* ends in .z). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

*Unpack* returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

a file with the unpacked name already exists

if the unpacked file cannot be created

**PASTE(1)****NAME**

**paste** - merge same lines of several files or subsequent lines of one file

**SYNOPSIS**

```
paste file1 file2...
paste -dlist file1 file2...
paste -s [-dlist] file1 file2...
```

**DESCRIPTION**

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2* and so on. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). It is the counterpart of *cat(1)* which concatenates vertically, that is, one file after the other. In the last form above, *paste* subsumes the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the tab character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if *-* is used in place of a file name.

The meanings of the options are:

- d** Without this option, the new-line characters of each but the last file (or last line in case of the **-s** option) are replaced by a tab character. This option allows replacing the tab character by one or more alternate characters (see below)
- list** One or more characters immediately following **-d** replace the default tab as the line concatenation character. The list is used circularly, that is, when exhausted, it is reused. In parallel merging (that is, no **-s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), **\\\** (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (for example, to get one backslash, use **-d“\\\\\\”**)
- s** Merge subsequent lines rather than one from each input file. Use tab for concatenation, unless a *list* is specified with **-d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line
- May be used in place of any file name, to read a line from the standard input. (There is no prompting)

**EXAMPLES**

<b>ls   paste -d“” -</b>	list directory in one column
<b>ls   paste -----</b>	list directory in four columns
<b>paste -s -d“\\t\\n” file</b>	combine pairs of lines into lines

**SEE ALSO**

**grep(1)**, **cut(1)**,  
**pr(1)**: **pr -t -m...** works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

**PASTE(1)****DIAGNOSTICS****line too long**

Output lines are restricted to 511 characters

**too many files**Except for -s option, no more than 12 input files  
may be specified

**PLOT(1)****NAME**

**plot** - plot window or screen  
**splot** - invoke plot program in background

**SYNOPSIS**

**plot** file  
**splot** file

**DESCRIPTION**

*plot* produces a file containing the image of the window or similar object, identified by *file*, on its standard output. The output data is preceded by a 512-byte header of format given in *usr/include/plot.h*.

*splot* makes a *system(3)* library routine call with the command string:  
*plot* *file* | *lpp&*.

**FILES**

*/usr/include/plot.h*

**SEE ALSO**

*lpp(1)*, *system(3)*, *lplota(3)*, *lplotb(3)*, *plot(5)*.



**PRIME(1M)****NAME**

**prime** - copy filestore issue disc to fixed disc

**SYNOPSIS**

**prime**

**DESCRIPTION**

*prime* copies all the files on a filestore issue floppy disc to the fixed disc, creating directories as necessary and overwriting any existing files of the same name.



**PRINTERR(1M)****NAME**

*printerr* - error reporting daemon

**SYNOPSIS**

/etc/printerr

**DESCRIPTION**

*printerr* collects error records by reading /dev/err and displays them, either on the screen accompanied by a bleep, or if WMS is running, in an error window opened by *printerr*. Only the superuser can start *printerr* and only one *printerr* daemon can be active at any time.

**FILES**

/dev/err Source of error records

**SEE ALSO**

*err*(4).



**PRS(1)****NAME**

**prs** - print an SCCS file

**SYNOPSIS**

**prs [-d[dataspec]] [-r[SID]] [-e [-l] [-a]] files**

**DESCRIPTION**

**prs** prints, on the standard output, parts or all of an SCCS file (see *sccsfile(5)*) in a user supplied format. If a directory is named, **prs** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.), and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

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

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

- |                     |   |
|---------------------|---|
| <b>-d[dataspec]</b> | Used to specify the output data specification. The <i>dataspec</i> is a string consisting of SCCS file <i>data keywords</i> (see <i>DATA KEYWORDS</i> ) interspersed with optional user supplied text   |
| <b>-r[SID]</b>      | Used to specify the <i>SCCS IDentification</i> (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed  |
| <b>-e</b>           | Requests information for all deltas created <i>earlier</i> than and including the delta designated via the <b>-r</b> keyletter  |
| <b>-l</b>           | Requests information for all deltas created <i>later</i> than and including the delta designated via the <b>-r</b> keyletter  |
| <b>-a</b>           | Requests printing of information for both removed, that is, delta type = <i>R</i> , (see <i>rmdel(1)</i> ) and existing, that is, delta type = <i>D</i> , deltas. If the <b>-a</b> keyletter is not specified, information for existing deltas only is provided |

**DATA KEYWORDS**

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile(5)*) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by **prs** consists of:

- 1 The user supplied text
- 2 Appropriate values (extracted from the SCCS file) substituted for the recognised data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return

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

**PRS(1)**

**Table 1**  
SCCS File Data Keywords

<i>Keyword</i>	<i>Data Item</i>	<i>File Section</i>	<i>Value</i>	<i>Format</i>
:Dt:	<i>Delta information</i>	<i>Delta Table</i>	<i>See below*</i>	S
:DL:	Delta line statistics	"	:Li:/:Ld:/	S
			:Lu:	
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	D or R	S
:I:	SCCS ID string (SID)	"	:R.:L.:B.:	S
			:S:	
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/	S
			:Dd:	
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm::	S
			:Ts:	
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/	S
			:Dg:	
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	yes or no	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	yes or no	S
:BF:	Branch flag	"	yes or no	S
:J:	Joint edit flag	"	yes or no	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	yes or no	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of what(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of what(1) string	N/A	:Z::Y: :M:	S
			:I: :Z:	
:Z:	what(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

\* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

**PRS(1)****EXAMPLES**

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

may produce on the standard output:

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

**xyz**

**131**

**abc**

**prs -d "Newest delta for pgm :M:I: Created :D:By:P;" -r s,file**

may produce on the standard output:

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

**As a special case:**

**prs s,file**

may produce on the standard output:

**D 1.1 77/12/1 00:00:00 cas 1 000000/000000/00000**

**MRs:**

**bl78-12345**

**bl79-54321**

**COMMENTS:**

**this is the comment line for s.file initial delta**

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

**FILES**

**/tmp/pr?????**

**SEE ALSO**

***admin(1), delta(1), get(1), help(1), sccsfile(5).***

**Bonanni, L.E. and Salemi, C.A., *Source Code Control System User's Guide*.**

**DIAGNOSTICS**

**Use *help(1)* for explanations.**



**PWCK(1M)****NAME**

**pwck, grpck - password/group file checkers**

**SYNOPSIS**

**pwck [file]**

**grpck [file]**

**DESCRIPTION**

*pwck* scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The criteria for determining a valid login name are taken from *Setting up UNIX*. The default password file is */etc/passwd*.

*grpck* verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default file is */etc/group*.

**FILES**

*/etc/group*

*/etc/passwd*

**SEE ALSO**

*group(5), passwd(5).*

*Setting up UNIX*

**DIAGNOSTICS**

Group entries in */etc/group* with no login names are flagged.



**RANLIB(1)****NAME**

**ranlib** - update symbol definitions for archive file(s)

**SYNOPSIS**

**ranlib lib1[lib2 lib3...]**

**DESCRIPTION**

*ranlib* creates/updates the symbol definition table (*\_\_.SYMDEF*) for the library archive format file(s) specified. It does this using the command *ar* (which would also be used to create/maintain the library). This command can be used when the symbol definitions are found to be out of date with the contents of the library.

**SEE ALSO**

*ar(1),ar(5),ld(1),nm(1).*



**REGCMP(1)****NAME**

*regcmp* - regular expression compile

**SYNOPSIS**

*regcmp* [ - ] files

**DESCRIPTION**

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

**EXAMPLES**

```
name "([A-Za-z][A-Za-z0-9]*)$0"
telno "\(\{0,1\}\{[2-9]\{01\}[1-9]\}\{0\}\{0,1\}**"
      "\{[2-9]\{0-9\}\{2\}\}\$\{[-]\{0,1\}\"
      "\{[0-9]\{4\}\}\{2\}"
```

In the program C that uses the *regcmp* output,

*regex(telno,line,area,exch,rest)*

will apply the regular expression named *telno* to *line*.

**SEE ALSO**

*regex(3X)*.



**RMDEL(1)****NAME**

*rmdel* - remove a delta from an SCCS file

**SYNOPSIS**

*rmdel* - rSID files

**DESCRIPTION**

*rmdel* removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must not be that of a version being edited for the purpose of making a delta (for example if a p-file (see *get(1)*) exists for the named SCCS file, the specified must *not* appear in any entry of the p-file).

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

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are:

- 1 If you make a delta you can remove it
- 2 If you own the file and directory you can remove a delta

**FILES**

x-file (see *delta(1)*)  
z-file (see *delta(1)*)

**SEE ALSO**

*delta(1), get(1), help(1), prs(1), sccsfile(5)*.

Bonanni, L.E. and Salemi; C.A., *Source Code Control System User's Guide*.

**DIAGNOSTICS**

Use *help(1)* for explanations.



**RSTTY(1)****NAME**

**rstty** - create a process group to run a control terminal on the RS232C interface

**SYNOPSIS**

**rstty** *t*

**DESCRIPTION**

*rstty* creates a separate process group to handle the device */dev/rstty* as a control terminal. This process group starts a shell process to handle the information sent from the remote machine designated */dev/rstty*.

The parameter takes a value 0 to 6 and specifies properties of the teletype device attached to the RS232C interface. The terminal types represented by these values are defined in *getty(8)*.

*rstty* terminates immediately leaving two processes associated with the control terminal on the RS232C interface. The PERQ console may then use the window or console.

When no longer required, the process pair associated with the device */dev/rstty* may be terminated using *kill*. Both processes may be terminated using a single kill command of the form

**kill** *pid*

where *pid* = the process number associated with *rstty*; determined by using *ps a* -this is because the process traps the signals SIGHUP and terminates the second (child) process. The command

**kill -1** *pid*

has an identical effect.

The process pair required for a control terminal on the RS232C interface may also be created at system initialisation time see *ttys(5)* and *init(8)*.

Note that only one process at a time can have the RS232 open.

**SEE ALSO**

*ps(1)*, *rstty(4)*, *ttys(5)*, *getty(8)*, *init(8)*.



**SACT(1)****NAME**

*sact* - print current SCCS file editing activity

**SYNOPSIS**

*sact* files

**DESCRIPTION**

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

The output for each named file consists of five fields separated by spaces:

- |         |   |
|---------|---|
| Field 1 | Specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta |
| Field 2 | Specifies the SID for the new delta to be created   |
| Field 3 | Contains the logname of the user who will make the delta (that is, executed a <i>get</i> for editing)                   |
| Field 4 | contains the date that <i>get -e</i> was executed   |
| Field 5 | contains the time that <i>get -e</i> was executed   |

**SEE ALSO**

*delta(1)*, *get(1)*, *unget(1)*.

**DIAGNOSTICS**

Use *help(1)* for explanations.



**SCCSDIFF(1)****NAME**

sccsdiff - compare two versions of an SCCS file

**SYNOPSIS**

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

**DESCRIPTION**

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

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

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

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

**FILES**

/tmp/get????? Temporary files

**SEE ALSO**

*bdiff(1)*, *get(1)*, *help(1)*, *pr(1)*.

Bonanni, L.E. and Salemi, C.A., *Source Code Control System User's Guide*

**DIAGNOSTIC**

**file:** No differences is output if the two versions are the same.

Use *help(1)* for explanations.



**SETPF(1)****NAME**

**setpf** - sets/returns strings assigned to the function keys

**SYNOPSIS**

**setpf [pf<n> = <string>,..]**

**DESCRIPTION**

*setpf* with no arguments returns the current settings of the function keys in the form:

PF1 = "<string>"

PF2 = "<string>"

PF3 = "<string>"

PF4 = "<string>"

*setpf* with arguments sets the PF keys. An argument is of the form:

pf<n> = <string>

where <n> is the PF key number (1, 2, 3 or 4) and <string> is a Cstring with C escape sequences.

If specifying a string with characters normally interpreted by the shell, enclose the arguments in single quotes. For example:

'pf1 = cd /usr/john'

The total length of the string cannot exceed 80 characters.



**SDB(1)****NAME**

*sdb* - symbolic debugger

**SYNOPSIS**

*sdb* [*objectfile* [*corefile* [*directory*]])

**DESCRIPTION**

*sdb* is a symbolic debugger for use with C, Fortran 77 or mixed C and Fortran 77 programs. *sdb* enables you to examine the files making up the program and provides a controlled environment for executing the program in parts and examining the state of the program following execution.

*Objectfile* is an executable program file compiled with the *-g* option (debug) set. The default is *a.out*. *Corefile* is assumed to be a core image file produced after executing object file. Whenever a program does not terminate normally a core image file is produced which is available to *SDB*. The default for core file is *core*. The core file need not be present. *Directory* specifies the directory *sdb* is to search for the source files of *objectfile*.

*sdb* maintains the idea of a current line, current procedure and a current file. If a core file exists, *sdb* initially sets the current line, procedure and file to the line, procedure and file containing the source statement at which the process terminated or stopped. If there is no core file, *sdb* initially sets the current line and file to the first line in *main()*. Commands to examine source files refer to the current line and may also change the current line, procedure or file.

*sdb* also maintains the idea of a line number. The line number is always relative to the beginning of the file. Lines in the source program may be referred to as *filename:number* or *procedure:number*; in both cases number is relative to the beginning of the file. If no procedure or filename is given then *sdb* assumes the current file is to be used. If no number is given, *sdb* assumes the first line of the named procedure or file is to be used.

*General sdb commands*

Commands to control *sdb* are as follows:

<b>h</b>	Help. Displays a list and brief explanation of all the <i>sdb</i> commands
<b>! command</b>	Passes <i>command</i> to the shell for execution
<b>q</b>	Quits <i>sdb</i> and returns to the shell
<b>v</b>	Prints the <i>sdb</i> version number
<b>B</b>	Prints a list of the currently active breakpoints
<b>D</b>	Deletes all breakpoints

*Examining source files*

The commands for examining source files are:

<b>e procedure</b>	
<b>e filename</b>	Sets the current file to the file containing <i>procedure</i> or to the <i>filename</i> . If no procedure ... filename is given this command reports the current procedure and file names
<b>p</b>	Prints the current line
<b>z</b>	Prints the current line followed by the next 9 lines. Sets the current line to the last line printed
<b>^Z or ^D</b>	Scroll. Prints the next 20 lines and sets the current line to the last line printed
<b>w</b>	Window. Prints the 20 lines around the current line

**SDB(1)**

<i>number</i>	Sets the current line to the given line number and prints the new current line
<i>count</i> +	Moves the current line forward by <i>count</i> lines and prints the new current line
<i>count</i> -	Moves the current line back by <i>count</i> lines and prints the new current line
/ <i>regular expression</i> /	Searches forward from the current line for a line containing a string matching <i>regular expression</i> . The last / may be omitted
? <i>regular expression</i> ?	Searches backward from the current line for a line containing a string matching <i>regular expression</i> . The last ? may be omitted

*Executing the program under sdb control*

You can run the program from within *sdb* and commands allow you to execute the program in stages and to examine data values at certain stages in the program.

Names of variables are written just as they appear in the C or Fortran 77 program. Remember that F77 variables called by a C procedure must have a trailing underscore added to their names as the F77 compiler adds a trailing underscore to all variable 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 the default. Structure members may be referred to by *variable.member*, pointers to structure members may be referred to by *variable->member* and array elements may be referred to by *variable[number]*. Combinations of these forms may be used.

Variables may also be specified by their addresses. All forms of integer constants valid in C may be used so addresses may be given in decimal, octal or hexadecimal.

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

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

**SDB(1)***linenumber b commands*

Sets a breakpoint at the given *linenumber*. Instead of a line number you can give a procedure name and a breakpoint is placed at the first line of the procedure even if it was not compiled with the debug flag.

If no line number or procedure is given, the breakpoint is placed at the current line.

When the breakpoint is encountered, any *commands* are executed and then the program continues. Multiple commands may be specified by separating them with semicolons. Valid commands are any of these *sdb* commands.

If *commands* are not specified execution stops just before the break and control is returned to *sdb*.

*linenumber d*

Deletes the breakpoint at the given line. If no linenumber is given then the breakpoints are deleted interactively. Each breakpoint location is printed out, one at a time. *sdb* waits for input from the standard input for each breakpoint. Input y or d to delete the breakpoint, anything else to keep it.

*linenumber a*

Announces when the program reaches *linenumber*. This is nearly the same as the break command except that the given line is printed out.

If *linenumber* is of the form *proc:number*, the command effectively does a *linenumber b l*. If *linenumber* is of the form *proc:*, the command effectively does a *proc: b T*.

*linenumber c count*  
*linenumber C count*

Continues running the program after a breakpoint or interrupt. *Count*, if given, specifies the number of breakpoints to be ignored. *C* continues with the signal which causes the program to stop and *c* ignores it. If a line number is specified then a temporary breakpoint is placed at the line specified. The breakpoint is deleted after the break.

*linenumber g count*

Continues after a breakpoint. Execution resumes from the given *linenumber*. *Count*, if given, specifies the number of breakpoints to be ignored.

*count s*  
*count S*

Runs one step of the program. *Count* specifies the number of program lines in the step. If *count* is not given the program runs for one line. *S* steps over subroutine calls. Performance of *s* is very slow.

*variable\$mcoun*  
*variable\$Mcount*  
*address:mcount*  
*address:Mcount*

Executes the program in single steps monitoring through count lines until the value of *variable* or the value at *address* changes. *address* is the address of a 4 byte memory location and may be specified as a decimal, octal or hex number. Both commands display the source line where the change occurs and the line number of that line. *M* also displays all source lines as they are executed from the current line to the line where the change occurs.

**SDB(1)**

If *count* is omitted, program executes till a breakpoint is encountered or to the end. If at any time a breakpoint is encountered one of two things can happen:

- 1 If breakpoint has no command list then monitor command terminates and control returns to *sdb*
- 2 If breakpoint has a command list, then commands are executed and monitoring continues

Warning: No address validation takes place upon the address specified.

Note: Performance of *m*, like *s*, is very slow

*procedure (args)*  
*procedure (args)/m*

Executes the named procedure with the given arguments. Arguments may be integer, character, double or string constants or names of variables accessible from the current procedure. If you wish to call functions or subroutines from *sdb* then the program must be linked with the *-lg* flag. Remember that F77 subroutines called by C must have a trailing underscore added to their F77 name

The first form of the command executes the procedure and if successful terminates with the message:

*procedure returned normally*

The second form of the command causes the value returned by the procedure to be printed according to format *m*. Format may be:

- c character
- d decimal
- u unsigned
- o octal
- x hexadecimal
- f floating point
- g double
- p pointer

If no format is given it defaults to d

**k**

*Examining data values*

*variable/lm*

Prints the value of *variable* according to length *l* and format *m*. If *l* and *m* are omitted, *sdb* chooses a length and format suitable for the variables type as declared in the program.

The length specifiers are:

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

*number* string length for formats s and a

**SDB(1)**

Formats are:

- c** character
- d** decimal
- u** decimal, unsigned
- o** octal
- x** hexadecimal
- C** F77 complex
- f** 32 bit single precision floating point
- g** 64 bit double precision floating point
- s** Assumes *variable* is a string pointer and prints characters starting at the address pointed to by the variable
- a** Prints character starting at address of variable
- p** Pointer to procedure

The length specifiers are only effective with the formats **d**, **u**, **o** and **x**. If one of these formats is specified and *l* is omitted, the length defaults to *l*. If a numeric length specifier is used for formats **s** or **a** then that many characters are printed. With no length specifier 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, both variables local to the current procedure and global (common for F77) variables are matched, while if a procedure name is specified then only variables local to that procedure are matched. To match only global variables (or blank common for F77), the form :*pattern* is used. The name of a common block may be specified instead of a procedure name for F77 programs.

*variable=lm*  
*linenumber=lm*  
*number=lm*

Prints the address of *variable* or *linenumber* or the value of *number* in the format specified by *lm*. If no format is given *sdb* assumes *lx*.

*variable!value*

The last variant of this command provides a convenient way to convert between decimal, octal and hexadecimal.

"*string*

Sets *variable* to the given *value*. *Value* may be a number, character, constant or a variable. If *variable* is of type float or double, *value* may also be a floating constant

Prints the given *string*. This command is useful for inclusion in lists of commands supplied to the break command **b**.

For example, if following the break you want to print the value of a certain variable you could use the string command to print the name of the variable first. The command would be:

*linenumber b " variable name : variable/lx*

## SDB(1)

The following commands may be used following a break in a program running under *sdb* or may equally be used to examine a program terminated unusually leaving a core file for *sdb* to look at:

- |          |   |
|----------|---|
| I        | Prints the last line executed   |
| T        | Prints a trace of the top of stack  |
| t        | Prints a stack trace of entire stack  |
| new-line | If the previous command printed a line this advances the current line by 1 line and prints the new current line. If the previous command displayed a core location then displays the next core location |

## FILES

a.out  
core

## SEE ALSO

*a.out(5)*, *core(5)*

## DIAGNOSTICS

self explanatory

## BUGS

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

Arrays must be of one dimension and of zero origin to be correctly addressed by *sdb*.

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

*sdb* cannot identify F77 subprograms with alternate entry points.

Therefore:

- 1 Entry points cannot be from *sdb* (entry may be made through the main entry)
- 2 Trace commands do not show entry points called (shows the subprogram as being called)
- 3 Parameters passed from entry points may not be examined in *sdb*

**SHUTDOWN(1M)****NAME**

Shutdown - terminate all processing

**SYNOPSIS**

/etc/shutdown

**DESCRIPTION**

Only the superuser can use *shutdown*. The command is functionally equivalent to:

**sync  
kill (-1, 1)**

The system shuts down all processing and reverts to single user mode.

**SEE ALSO**

*sync(1M), kill(1)*.



**SLAVEFT(1)****NAME**

*slaveft* - provide a service response on an Open Systems LAN to a call from *mftp* on another PNX PERQ, or from PERQFTP on a POS PERQ

**SYNOPSIS**

not applicable

**DESCRIPTION**

*slaveft* does not require any action by the user of the calling machine. When the user opens a connection to a remote machine using the *mftp* master calling program facility (see *mftp*), *slaveft* acts as the responder program or slave. It forks a child process of itself in the calling machine which remains resident in that machine until the connection is broken. The fork is created by the system utilities *execl* and the *ioctl* command **ECMATASUM**. **ECMATASUM** will only fork the child *slaveft* process if there is no service with the same TSEL-ID currently available. For details of TSEL-ID see the publication *ICL PERQ: Establishing IPA under PNX*. The check of the TSEL-ID is made when a connection is first made; after opening a file; if a network failure occurs; and before the final exit when it will almost certainly not be necessary.

Each machine that is required to make a response on a network must have a copy of *slaveft* on its file store.

*slaveft* must be loaded originally by the user of responder machine by copying it from floppy to users' directory */etc/rc*. The user inserts */etc/slaveft&*, the machine will then respond to any other user from an *mftp* PNX PERQ or from PERQFTP on a POS PERQ.

The user running *mftp* in the calling machine will supply a username and the current directory. No file manipulation will be permitted unless the *mftp* calling program supplies a valid username and password combination. Files will only be transferred if the appropriate access rights are satisfied.

For details of PERQFTP see the publication *ICL PERQ: System Software Reference (POS) (R10101/00, Edition 4)*.

**FILES**

*/etc/rc*  
*/fork/exec*  
*/ioctl/ECMATASUM*  
*/etc/slaveft&*



**SPY(1)**

**NAME**

## **spy - a screen-oriented editor for PNX**

## **SYNOPSIS**

```
SPYOPTIONS=-eflpqrstw; export SPYOPTIONS
spy [ -eflpqrstw ] [ filename ... ]
```

## **DESCRIPTION**

*spy* is a screen editor that uses the pointing abilities of the puck to provide a fast, convenient, and easy to learn editing tool. A full description appears in *Guide to PNX*. Only a summary of the special characters is given here. To provide continuity between the standard *tty* interface and *spy*, the usual editing actions of the characters kill, erase (**DEL** and **OOPS** by default), **^W**, and **^R** are provided. In addition linefeed, **LF**, inserts a new line and copies the indentation of the previous line.

The search string matches patterns expressed using the following meta-language symbols (see [regex\(3\)](#)). An entirely empty search string matches the inter-character gap.

- |            |   |
|------------|---|
| .          | matches any character except newline  |
| []         | encloses the classes of characters which may be matched (for example, [0-9a-zA-Z] matches any alphanumeric). In this context, ., *, ^ and [ have no special meaning. \ is special only when used to denote a new-line with \n. ^ is special only at the beginning of the enclosed string when it causes the class to match any character except the remaining characters in the string. - may be used to indicate a range of consecutive characters, but loses its special meaning if it occurs first (after any initial ^) or last in the string. ] does not terminate such a string when it is the first characters within it (after any initial ^) |
| LF         | and RETURN and \n match a new-line  |
| *          | at the beginning of the string matches the beginning of a line  |
| \$         | at the end of the string matches the end of a line, excluding its new-line  |
| (...)      | may be used to group characters into a regular expression   |
| (...)\${i} | the value of the enclosed regular expression is saved and may be called up by the metacharacters \\$i in the replace string   |
| *          | applies the preceding regular expression zero or more times and is equivalent to {0,}   |
| +          | applies the preceding regular expression one or more times and is equivalent to {1,}  |
| {m,u}      | denotes the minimum and maximum times the preceding regular expression may be applied; u must be less than 256  |
| {m}        | means match exactly m times   |
| {m,}       | means match m or more times   |
| \          | removes any special meaning from the next character; see also the -p flag   |
| &          | in the replace string is replaced by the whole selection  |
| \i         | is replaced by the part of the selection which matches (...)\${i} in the search string. (If there is no match, this is null.)   |

## SPY(1)

Initial options can be given on the PNX command by preceding them with a -. Several filenames can be given to edit. If the name SPYOPTIONS is in the environment, its value will be used as an extra first argument. The default options are the opposite of the ones that can be given on the command, which are listed below.

- t No tabbing. Tab characters (^I) are no longer expanded into 1 to 7 places in the display
- f Full font. Control characters including new-line are displayed according to the font, instead of in an inverted video
- q Quick. Instead of showing changes to the screen line by line they are shown only when they have all been done
- s Silence. No bells are sounded
- p No patterns. No special metacharacters in the search and replace strings are recognised
- r Raw mode. The characters DEL, OOPS, RETURN, LF, ^W and ^R have their special meaning removed. LF is needed to generate a new line
- w Wrapping. Search will wrap past the start or end of file to find a match
- l No labels. The meaning of the puck buttons is no longer shown in the function box
- e The next argument is a filename. This allows filenames to begin with -

A *kill -15* (SIGTERM) or *-20* (when the window size is changed) signal redraws the window, if it is still large enough. The sub-windows become evenly spaced. Other signals stop the editor when any current command finishes. A further signal attempts to stop immediately. Sub-windows that have not been written back have .hup files created, if possible.

The standard input for the editor must be defaulted to the window it is to run in. If a proportional font exists for the window, it will be drawn with the characters evenly spaced.

## FILES

/usr/pub/asciichars  
/usr/lib/cursors/spy/HELP  
/usr/lib/cursors/spy/SPYFONT  
/usr/lib/cursors/spy/HE\_CURS  
/usr/lib/cursors/spy/CB\_CURS  
/usr/lib/cursors/spy/TX\_CURS  
/usr/lib/cursors/spy/TB\_CURS  
/usr/lib/cursors/spy/SC\_CURS  
/usr/lib/cursors/spy/HANDB

## SEE ALSO

*Guide to PNX*

## DIAGNOSTICS

Internal consistency failures will produce a message on a black background. After 5 seconds the editor will continue. The message will say if the file at fault can be usefully written to disc in an attempt to save its contents.

## BUGS

A fault in the microcode version V1.6 results in calls to *realloc* corrupting the data that is moved. Some of the mess that ensues can be detected by the editor and where possible some recovery is made.

**SRS(1)****NAME**

**srs** - set RS232C device options

**SYNOPSIS**

**srs [option ...]**

**DESCRIPTION**

**srs** sets certain I/O options on the current device. With no argument, it reports the current settings of the options. The option strings are selected from the following set:

<b>rsa</b>	set/report the I/O options of <i>/dev/rsa</i>
<b>rsb</b>	set/report the I/O options of <i>/dev/rsb</i>
<b>rs</b>	set/report the I/O options of <i>/dev/rs</i>
<b>reset</b>	put device into default state
<b>even</b>	set even parity
<b>odd</b>	set odd parity
<b>any</b>	set any parity
<b>noparity</b>	set no parity
<b>-even</b>	if even parity set, set no parity
<b>-odd</b>	if any parity set, set odd parity
<b>-parity</b>	if odd parity set, set no parity
<b>-any</b>	if any parity, set even parity
<b>raw</b>	set raw mode
<b>-raw</b>	unset raw mode
<b>en</b>	enable input
<b>-en</b>	disable input
<b>br5</b>	set the number of bits per received character.
<b>br6</b>	This includes the parity bit if parity is specified.
<b>br7</b>	
<b>br8</b>	
<b>bt5,</b>	set the number of bits per transmitted character.
<b>bt6</b>	This includes the parity bit if parity is specified
<b>bt7</b>	
<b>bt8</b>	
<b>st1</b>	set the number of stop bits to 1, 1.5, or 2
<b>st2</b>	respectively. There is always one start bit.
<b>st3</b>	
<b>Ext</b>	set External Clocking Source.
<b>100, 150,</b>	set the baud rate to the specified value.
<b>300, 600,</b>	Transmit and receive baud rates are set to the
<b>1200, 2400</b>	same value.
<b>4800, 9600</b>	
<b>19200</b>	set the baud rate to 19.200 (PERQ 2 only)
<b>term &lt;c&gt;</b>	set the termination character to the first character of <c>. <c> can be of the form:  ^<X> : interpreted as "control <X>" \<X> : interpreted as the standard C escape sequences
<b>graphic characters</b>	Since both '^' and '\' are meaningful to the shell, it is safest to enclose <c> in single quotes, '...'.  

**SRS(1)**

In raw mode, the RS232C driver returns all characters, up to the maximum specified, that are available. As soon as there are no more characters available, the *read* completes, even if fewer characters have been read than specified in the *read* request. If there are no characters available the *read* waits. A *read* must return at least one character. As soon as a character is available, the *read* returns. At low line speeds it is likely that a *read* will read characters faster than they arrive and empty the buffer.

In raw mode, any termination character set is not recognised so there is no way of signalling the remote machine that termination has finished.

The default settings of these parameters are:

rs232 input enabled

9600 baud

transmit 8 bits, receive 8 bits, 1.5 stop bits, no parity

termination character = ^C (ETX or hex 03)

Note: If both even and odd parity are set then this is interpreted as no parity.

**STTY(1)****NAME**

**stty** - set terminal options

**SYNOPSIS**

**stty** [option ...]

**DESCRIPTION**

**stty** sets certain I/O options on the current output terminal. With no argument, it reports the current setting of the options. The option strings are selected from the following set:

<b>raw</b>	raw mode input (no erase, kill, interrupt, quit, EOT; any parity bit (bit 7) is passed back but the control key does not set bit 7)
<b>-raw</b>	negate <b>raw</b> mode
<b>cooked</b>	same as <b>-raw</b>
<b>cbreak</b>	make each character available to <i>read(2)</i> as received; no erase and kill
<b>-cbreak</b>	make characters available to <i>read(2)</i> only when new-line is received
<b>echo</b>	echo back every character typed
<b>-echo</b>	do not echo characters
<b>lcase</b>	map upper case to lower case
<b>-lcase</b>	do not map case
<b>ek</b>	reset erase and kill characters back to normal <b>DEL</b> and <b>^U</b>
<b>erase</b> <i>c</i>	set erase character to <i>c</i> . <i>c</i> can be of the form '^X' which is interpreted as a 'control X'.
<b>kill</b> <i>c</i>	set kill character to <i>c</i> . '^X' works here also
<b>width</b> <i>n</i>	set the page width to <i>n</i> characters for automatic line folding
<b>length</b> <i>n</i>	set the page length to <i>n</i> lines, so output may be paged. A length of 0 turns paging off completely
<b>indctl</b>	echo control-characters as '^<char+0141>'
<b>-indctl</b>	turn off <b>indctl</b> mode



**UNAME(1)****NAME**

**uname** - print name of current UNIX

**SYNOPSIS**

**uname [-snrva]**

**DESCRIPTION**

*uname* prints the current system name of UNIX on the standard output file. It is mainly useful to determine what system one is using. The options cause selected information returned by *uname(2)* to be printed:

- s print the system name (default)
- n print the nodename (the nodename may be a name that the system is known by to a communications network)
- r print the operating system release
- v print the operating system version
- a print all the above information

**SEE ALSO**

*uname(2)*.



**UNGET(1)****NAME**

**unget** - undo a previous get of an SCCS file

**SYNOPSIS**

**unget** [-rSID] [-s [-n] files

**DESCRIPTION**

*unget* undoes the effect of a *get -e* done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file:

- |              |  |
|--------------|--|
| <b>-rSID</b> | Uniquely identifies which delta is no longer intended. (This would have been specified by <i>get</i> as the "new delta"). The use of this keyletter is necessary only if two or more outstanding <i>gets</i> for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified <i>SID</i> is ambiguous, or if it is necessary and omitted on the command line |
| <b>-s</b>    | Suppresses the printout, on the standard output, of the intended delta's <i>SID</i>  |
| <b>-n</b>    | Causes the retention of the gotten file which would normally be removed from the current directory   |

**SEE ALSO**

*delta(1)*, *get(1)*, *sact(1)*

**DIAGNOSTIC**

Use *help(1)* for explanations.



**VAL(1)****NAME**

*val* - validate SCCS file

**SYNOPSIS**

*val* -  
*val* [-s] [-rSID] [-mname] [-ytype] files

**DESCRIPTION**

*val* determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* appear in any order. The arguments consist of keyletter arguments, which begin with a -, and named files.

*val* has a special argument, - which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

*val* generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line:

- |        |   |
|--------|---|
| -s     | The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line   |
| -rSID  | The argument value <i>SID</i> ( <i>SCCS IDentification String</i> ) is an SCCS delta number. A check is made to determine if the <i>SID</i> is ambiguous (for example, r is ambiguous because it physically does not exist but implies 1.1, 1.2, and so on, which may exist) or invalid (for example, r1.0 or r1.1.0 are invalid because neither case can exist as a valid delta number). If the <i>SID</i> is valid and not ambiguous, a check is made to determine if it actually exists. |
| -mname | The argument value <i>name</i> is compared with the SCCS %M% keyword in file.   |
| -ytype | The argument value <i>type</i> is compared with the SCCS %Y% keyword in file  |

The 8-bit code returned by *val* is a disjunction of the possible errors that is, it can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

- bit 0 = missing file argument
- bit 1 = unknown or duplicate keyletter argument
- bit 2 = corrupted SCCS file
- bit 3 = can't open file or file not SCCS
- bit 4 = *SID* is invalid or ambiguous
- bit 5 = *SID* does not exist
- bit 6 = %Y%, -y mismatch
- bit 7 = %M%, -m mismatch

Note that *val* processes two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned - a logical OR of the codes generated for each command line and file processed.



**WHAT(1)****NAME**

**what** - identify SCCS files

**SYNOPSIS**

**what** files

**DESCRIPTION**

**what** searches the given files for all occurrences of the pattern that *get(1)* substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ">", new-line, \, or null character. For example, if the C program in file f.c contains

```
char ident[] = "@(#) identification information";
```

and f.c is compiled to yield f.o and a.out, then the command

**what f.c f.o a.out**

will print

f.c:

identification information

f.o:

identification information

a.out:

identification information

**what** is intended to be used in conjunction with the command *get(1)*, which automatically inserts identifying information, but it can also be used where the information is inserted manually.

**SEE ALSO**

*get(1)*, *help(1)*.

**DIAGNOSTIC**

Use *help(1)* for explanations.

**BUGS**

It's possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.



**WHODO(1M)****NAME**

*whodo* - who is doing what

**SYNOPSIS**

*/etc/whodo*

**DESCRIPTION**

*whodo* produces merged, reformatted, and dated output from the *who(1)* and *ps(1)* commands.

**SEE ALSO**

*ps(1)*, *who(1)*.



**WINIT(1)****NAME**

winit - initialise the Window Management System

**SYNOPSIS**

winit

**DESCRIPTION**

This program initialises the WMS, and executes the Window Manager program *winman*. The system window appears at the top right-hand corner of the screen, and the keyboard and tablet are monitored by the operator interface program.

Window facilities, other than those of */dev/screen*, must not be used without the WMS being active. The *winit* command must not be given again while the WMS is active.

*winman* executes *splot* and *whelp* in response to Window Manager *plot* and *help* commands.

*WMS termination*

To reduce the chance of accidental termination, WMS ignores EOF (^Z) and interrupt (^C). The quit (^\\) key causes termination of all user windows. To terminate the WMS, delete the system window with the delete command and verify the deletion with a reply of **y** followed by RETURN to the prompt:

**terminate window manager: y or n**

in the system window.

Alternatively, select the system window and type:

**terminate**

followed by RETURN

**SEE ALSO**

*wcurs(5)*, *wprofile(5)*.



**WSETUP(1M)****NAME**

wsetup - creates the WMS window descriptors

**SYNOPSIS**

/etc/wetc/wsetup

**DESCRIPTION**

*wsetup* creates the window descriptors - the system window *wman* and the two help windows *-genwnd* and *comwnd* in directory */etc/wdev*, which are required by the Window Management System. The command file is executed automatically during WMS installation, and may also be used to recreate the descriptors if they are accidentally deleted.

**FILES**

/etc/wsetup, /etc/mkhelp, /etc/mkwman



**XARGS(1)****NAME**

xargs - construct argument list(s) and execute command

**SYNOPSIS**

xargs [flags] [command [initial-arguments]]

**DESCRIPTION**

xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

*Command*, which may be a shell file, is searched for, using one's \$PATH. If *command* is omitted, /bin/echo is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted; characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see -i flag). Flags -i, -l and -n determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (for example, -l vs. -n), the last flag has precedence. Flag values are:

**-l***number*      *Command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line unless the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted 1 is assumed. Option -x is forced

**-i***replstr*      Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option -x is also forced. {} is assumed for *replstr* if not specified

**-n***number*      Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters and for the last invocation if there are fewer than *number* arguments remaining. If option -x is also coded, each *number* arguments must fit in the *size* limitation, else xargs terminates execution

**-t**      Trace mode: the *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution

**XARGS(1)**

-p	Prompt mode: the user is asked whether to execute <i>command</i> each invocation. Trace mode (-t) is turned on to print the command instance to be executed, followed by a ?... prompt. A reply of y (optionally followed by anything will execute the command; anything else, including just a carriage return, skips that particular invocation of <i>command</i> )
-x	Causes <i>xargs</i> to terminate if any argument list would be greater than <i>size</i> characters; -x is forced by the options -i and -l. When neither of the options -i, l, or -n are coded, the length of all arguments must be within the <i>size</i> limit
-ssize	The maximum total size of each argument list is set to <i>size</i> characters; <i>size</i> must be a positive integer less than or equal to 470. If -s is not coded, 470 is taken as the default. Note that the character count for <i>size</i> includes one extra character for each argument and the count of characters in the command name
-eofstr	<i>Eofstr</i> is taken as the logical end-of-file string. Underbar (_) is assumed for the logical EOF string if -e is not coded. -e with no <i>eofstr</i> coded turns off the logical EOF string capability (underbar is taken literally). <i>xargs</i> reads standard input until either end-of-file or the logical EOF string is encountered

*xargs* will terminate if either it receives a return code of -1 from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh(1)*) with an appropriate value to avoid accidentally returning with -1.

**EXAMPLES**

The following will move all files from directory to directory , and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}  
1
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $* | xargs >>log  
2
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1) one at a time, or (2) many at a time.

- 1      ls | xargs -p -l ar r arch
- 2      ls | xargs -p -l | xargs ar r arch

The following will execute *diff(1)* with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff  
3
```

**DIAGNOSTICS**

Self explanatory.

**XREF(1)****NAME**

xref - cross reference for C programs

**SYNOPSIS**

xref [ file ... ]

**DESCRIPTION**

*xref* reads the named *files* or the standard input if no file is specified and prints a cross reference consisting of lines of the form

identifier      file-name      line-numbers ...

Function definition is indicated by a plus sign (+) preceding the line number.

**SEE ALSO**

*cref(1)*.



**IOCTL(2)****NAME**

*ioctl*, *stty*, *gtty* - control device or window

**SYNOPSIS**

```
#include <sgtty.h>           /* for control device */
ioctl(fildes,request,argp)
struct sgttyb *argp;
stty(fildes,argp)
struct sgttyb *argp;
gtty(fildes,argp)
struct sgttyb *argp
#include <sgwin.h>           /* for windows */
ioctl (fildes, request, argp)
union sgwinb * argp;
```

**DESCRIPTION**

*ioctl* performs a variety of functions on character special files (devices) and windows. The writeup of various devices in section 4 and of windows in *window(4)* describes how *ioctl* applies to them. *ioctl*, read in conjunction with *tty(4)*, describes control functions relevant to use of a window as a terminal.

For certain status setting and status inquiries about terminals, the functions *stty* and *gtty* are equivalent to:

```
ioctl(fildes,TIOCSETP,argp)
ioctl(fildes,TIOCSETP,argp)
```

respectively; see *tty(4)*.

The following two calls, however, apply to any open file:

```
ioctl(fildes,FIOCLEX,NULL)
ioctl(fildes,FIONCLEX,NULL)
```

The first causes the file to be closed automatically during a successful exec operation; the second reverses the effect of the first.

**SEE ALSO**

*stty(1)*, *exec(2)*, *tty(4)*, *window(4)*

**DIAGNOSTICS**

Zero is returned if the call was successful; -1 if the file descriptor does not refer to the type of file intended, or if the operation is not permitted for the window.

**BUGS**

Strictly speaking, since *ioctl* may be extended in different ways to devices with different properties, *argp* should have an open ended declaration like:

```
union {struct sgttyb ... ; ... } *argp;
```

The important thing is that the size is fixed by *sgttyb* or *sgwinb*.



**MKNOD(2)****NAME**

*mknod* - make a directory or a special file.

**SYNOPSIS**

```
mknod (name, mode, addr)
char *name;
```

**DESCRIPTION**

*mknod* creates a new file whose name is the null-terminated string pointed to by *name*. The mode of the new file (including directory and special file bits) is initialised from *mode*. (The protection part of the mode is modified by the process's mode mask; see *umask(2)*). The first block pointer of the i-node is initialised from *addr*. For ordinary files and directories *addr* is normally zero. In the case of a special file, *addr* specifies which special file.

An additional mode value signifies that the special file required is to be a window device. In this case the file must already exist, and should contain data defining the required window properties. The format of this data is defined by the structures in *wdesc.h* (see *wdesc(5)*).

*mknod* may only be invoked by the superuser, except when making a window description file when the file owner may invoke *mknod*.

**SEE ALSO**

*mkdir(1)*, *mknod(1)*, *filsys(5)*, *wdesc(5)*

**DIAGNOSTICS**

Zero is returned if the file has been made; -1 if the file already exists or if the user is not the superuser.



**PTRACE(2)****NAME**

*ptrace - process trace*

**SYNOPSIS**

```
int ptrace (request, pid, addr, data);
in request, pid, addr, data;
```

**DESCRIPTION**

*Ptrace* provides a means by which a parent process may control the execution of a child process. Its primary use is for the implementation of breakpoint debugging; see *sdb(1)*. The child process behaves normally until it encounters a signal (see *signal(2)* for the list), at which time it enters a stopped state and its parent is notified via *wait(2)*. When the child is in the stopped state, its parent can examine and modify its "core image" using *ptrace*. Also, the parent can cause the child either to terminate or continue, with the possibility of ignoring the signal that caused it to stop.

The *request* argument determines the precise action to be taken by *ptrace* and is one of the following:

- 0** This request must be issued by the child process if it is to be traced by its parent. It turns on the child's trace flag that stipulates that the child should be left in a stopped state upon receipt of a signal rather than the state specified by *func*; see *signal(2)*. The *pid*, *addr*, and *data* arguments are ignored, and a return value is not defined for this request. Peculiar results will ensue if the parent does not expect to trace the child

The remainder of the requests can only be used by the parent process. For each, *pid* is the process ID of the child. The child must be in a stopped state before these requests are made.

- 1, 2** With these requests, the word at location *addr* in the address space of the child is returned to the parent process. If I and D space are separated (as on PDP-11s), request 1 returns a word from I space, and request 2 returns a word from D space. If I and D space are not separated (as on the VAX-11/780), either request 1 or request 2 may be used with equal results. The *data* argument is ignored. These two requests will fail if *addr* is not the start address of a word, in which case a value of -1 is returned to the parent process and the parent's *errno* is set to EIO
- 3** With this request, the word at location *addr* in the child's USER area in the system's address space (see <sys/user.h>) is returned to the parent process. Addresses in this area range from 0 to 1024 on the PDP-11s and 0 to 2048 on the VAX. The *data* argument is ignored. This request will fail if *addr* is not the start address of a word or is outside the USER area, in which case a value of -1 is returned to the parent process and the parent's *errno* is set to EIO
- 4, 5** With these requests, the value given by the *data* argument is written into the address space of the child at location *addr*. If I and D space are separated (as on PDP-11s), request 4 writes a word into I space, and request 5 writes a word into D space. If I and D space are not separated (as on the VAX), either request 4 or request 5 may be used with equal results. Upon successful completion, the value written into the address space of the child is returned to the parent. These two requests will fail if *addr* is a location in a pure procedure space and another process is executing in that space, or *addr* is not the start address of a word. Upon failure a value of -1 is returned to the parent process and the parent's *errno* is set to EIO

**PTRACE(2)**

- 6 With this request, a few entries in the child's USER area can be written. *cd2Data* gives the value that is to be written and *addr* is the location of the entry. The few entries that can be written are:  
 the general registers (that is, registers 0-7 on PDP-11s, and registers 0-15 on the VAX)  
 the floating point status register and six floating point registers on PDP-11s  
 certain bits of the Processor Status Word on PDP-11s (that is, bits 0-4, and 8-11)  
 certain bits of the Processor Status Longword on the VAX (that is, bits 0-7, 16-20, and 30-31)
- 7 This request causes the child to resume execution. If the *data* argument is 0, all pending signals including the one that caused the child to stop are cancelled before it resumes execution. If the *data* argument is a valid signal number, the child resumes execution as if it had incurred that signal and any other pending signals are cancelled. The *addr* argument must be equal to 1 for this request. Upon successful completion, the value of *data* is returned to the parent. This request will fail if *data* is not 0 or a valid signal number, in which case a value of -1 is returned to the parent process and the parent's *errno* is set to EIO
- 8 This request causes the child to terminate with the same consequences as *exit(2)*
- 9 This request sets the trace bit in the Processor Status Word of the child (that is, bit 4 on PDP-11s; bit 30 on the VAX) and then executes the same steps as listed above for request 7. The trace bit causes an interrupt upon completion of one machine instruction. This effectively allows single stepping of the child.  
 Note: the trace bit remains set after an interrupt on PDP-11s but is turned off after an interrupt on the VAX

PNX makes the following changes and additions to *ptrace*.

<i>Request number</i>	<i>Action</i>
0	Same as above
1 & 4	The address of I space value is as follows: MS 16 bits = function number as in namelist LS 16 bits = byte offset from start of function
2 & 5	If the address of D space is negative then the stack is accessed. Otherwise program static data is accessed
3	The maximum valid address is that of the last item in the user structure
6	The address is a register number 0-3 = general register number 4-5 = MS + LS 32 bits of floating point register 0 6-7 = MS + LS - floating point register 1 8-9 = MS + LS - floating point register 2 A-B = MS + LS - floating point register 3
7	Same as 7 above except that "Trap-On-Jump" mode is turned off

**PTRACE(2)**

<i>Request number</i>	<i>Action</i>
9	<p>Very different from 9 above. The PERQ cannot execute single machine instructions. This request executes as System 3 request 7 but does not turn off "Trap-On-Jump" mode.</p> <p>Trap-On-Jump mode works as follows. The program is executed as normal. Since the Trap-On-Jump flag is set in the user area, the C machine microcode interpreter monitors the instructions it is executing. If it detects an instruction involving a transfer of control (<i>jump, call, return or switch</i>) it executes that instruction, stops further execution and tells the kernel to send the SIGTOJ signal to that program.</p> <p>For conditional jump statements, SIGTOJ is only sent if the jump occurs.</p> <p>This mode is disabled if a system call is made. The mode is reenabled just before the return from the call</p>
10	<p>A new request. Designed for calling out of line procedures. Requests 7 and 9 do out of line jumps not calls.</p> <p><i>addr</i> is the address as in requests 1 and 4</p> <p>To forestall possible fraud, <i>ptrace</i> inhibits the set-user-id facility on subsequent <i>exec(2)</i> calls. If a traced process calls <i>exec</i>, it will stop before executing the first instruction of the new image showing signal <b>SIGTRAP</b>.</p>

**GENERAL ERRORS**

*Ptrace* will in general fail if one or more of the following are true:

*Request* is an illegal number. [EIO]

*Pid* identifies a child that does not exist or has not executed a *ptrace* with request 0. [ESRCH]

**SEE ALSO**

*sdb(1), exec(2), signal(2), wait(2).*



**SIGNAL(2)****NAME**

`signal - catch or ignore signals`

**SYNOPSIS**

```
#include <signal.h>
(*signal(sig, func))()
(func);
```

**DESCRIPTION**

A signal is generated by some abnormal event, initiated either by the user at a typewriter (quit, interrupt), by a program error (bus error, and so on), or by request of another program (kill). Normally all signals cause termination of the receiving process, but a `signal(2)` call allows them either to be ignored or to cause an interrupt to a specified location. Here is the list of signals with names as in the include file.

SIGHUP 1	hangup
SIGINT 2	interrupt
SIGQUIT 3*	quit
SIGILL 4*	illegal instruction (not reset when caught)
SIGTRAP 5*	used by kernel to inform <code>sdb(1)</code> that a breakpoint has been encountered in the process being debugged
SIGIOT 6*	IOT instruction
SIGEMT 7*	EMT instruction
SIGFPE 8*	floating point exception
SIGKILL 9	kill (cannot be caught or ignored)
SIGTOJ 10*	sent to a process whenever process executes a successful transfer of control ( <i>jump, call, return or switch</i> ) when process is run in Trap-on-jump mode (see <i>ptrace</i> )
SIGSEGV 11*	segmentation violation
SIGSYS 12*	bad argument to system call
SIGPIPE 13	write on a pipe or link with no one to read it
SIGALRM14	alarm clock
SIGTERM15	software termination signal
16	unassigned

The starred signals in the list above cause a core image if not caught or ignored. SIGTRAP and SIGTOJ are both used by `sdb` to implement debugging facilities so should not be used in user's programs as debugging a program that uses these signals would be difficult.

The following signal is for indicating Service Request for a device on GPIB (see *GPIB(4)*)

`SIGGPIB 19 SRQ on GPIB`

The default action is that it is ignored.

Three additional signals are for use with windows:

SIGWSTS 20	window status change
SIGWINP 21	window graphic input available
SIGHHELP 22	HELP key pressed

The default is that they are ignored.

These signals are sent to all processes which have an opened file descriptor to the window which has caused the signal.

If `func` is `SIG_DFL`, the default action for signal `sig` is reinstated; this default is termination, sometimes with a core image. If `func` is `SIG_IGN` the signal is ignored. Otherwise when the signal occurs `func` will be called with the signal number as argument. A return from the function will continue the process at the point it was interrupted. Except as indicated, a signal is reset to `SIG_DFL` after being caught. Thus if it is desired to catch every such signal, the catching routine must issue another `signal` call.

## SIGNAL(2)

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during a *read* or *write(2)* on a slow device (like a typewriter; but not a file); and during *pause* or *wait(2)*. When such a signal occurs, the saved user status is arranged in such a way that when return from the signal-catching takes place, it will appear that the system call returned an error status. The user's program may then, if it wishes, re-execute the call.

The value of *signal* is the previous (or initial) value of *func* for the particular signal.

After a *fork(2)* the child inherits all signals. *exec(2)* resets all caught signals to default action.

## SEE ALSO

*kill(1)*, *kill(2)*, *ptrace(2)*, *setjmp(3)*, *ioctl(2)*, *window(4)*, *screen(4)*.

## DIAGNOSTICS

The value (int)-1 is returned if the given signal is out of range.

## BUGS

If a repeated signal arrives before the last one can be reset, there is no chance to catch it. The type of specification of the routine and its *func* argument are problematical.

**WDBYTE(2)****NAME**

wdbyte - draw characters from a string to a window

**SYNOPSIS**

```
#include <wuser.h>
wdbyte (fildes,dbargp,clipp)
struct DBCtl *dbargp;
struct ClipCtl *clipp;
```

**DESCRIPTION**

This function draws characters from a string, using a specified font (set of character patterns), starting at a specified point in a window, until one of the following occurs:

- 1 the string is exhausted
- 2 a right hand margin is encountered (user specified)
- 3 a control character is encountered in the string (that is, character value, 0 to 31)

A clip rectangle may be specified as for *wrasop(2)*. Points which would be outside this area, and if the destination is the window, points which would be outside it, are not drawn, but the termination conditions above are evaluated nevertheless.

*Parameters*

fildes: /\*file descriptor of an open window file\*/

struct DBCtl {

short	DBX	;	/* Destination X co-ordinate */
short	DBByteOffset	;	/* Offset from start of string */
short	DBFunc	;	/* ROP function */
short	DBY	;	/* destination Y base coordinate */
short	DBMaxX	;	/* Max X Displacement */
short	DBMaxByte	;	/* Max Index to string */
char	*DBSrcString	;	/* source string */
int	*DBScreen	;	/* Address of screen (0 = window) */
struct	font *DBFont	;	/* Address of font (0 = default) */
short	DBDstInc	;	/* destination area width */

};

struct *ClipCtl* is defined in *wrasop(2)*. An address of zero denotes no additional clipping.

*Notes*

1. *DBX* is co-ordinate of left edge of first character to be drawn.
2. *DBY* is co-ordinate at which base line of characters is drawn.
3. *DBByteOffset* and *DBMaxByte* are character indexes within *DBSrcString*, which is a character address in process store.
4. *DBMaxX* is a maximum co-ordinate up to but not including which complete characters can be drawn.
5. If *DBScreen* is not zero, it specifies an address in process store: only in this case is *DBDstInc* relevant (see *wrasop* for restrictions on use of process store)

## WDBYTE(2)

6. If *DBFont* is not zero, it is taken to be the address of an area in process store representing a font (see *wrasop* for restrictions on use of process store). Fonts held on disc files may be mapped onto the following structure defined in */usr/include/wuserh*, although in practice the font data will continue past the end of the structure:

```
struct font_elem {  
    unsigned font_lno:6;          /* which line of the font data */  
    unsigned font_offset:10;       /* bit offset within the line */  
    unsigned font_width:16;        /* character width */  
};  
  
struct font {  
    short font_height;  
    short font_base;  
    struct font_elem font_char[0200]           /* character index  
                                                into font data */  
    int font_filler;                         /* align on 4 word  
                                                boundary */  
    short font_line[1];                      /* the font data */  
};
```

7. *DBX* and *DBByteOffset* are updated to specify the "next" values which would be used if the terminating condition had not occurred

### SEE ALSO

*wrasop(2)*, *wline(2)*, *rectangle(3)*, *window(4)*, *screen(4)*

### DIAGNOSTICS

Returned value	Meaning
0	successful (end of string, that is, <i>DBMaxByte</i> - <i>DBByteOffset</i> characters drawn)
-1	<i>fildes</i> does not refer to an open window file (or, see also <i>errno</i> under <i>intro(2)</i> )
1	a right hand margin <i>DBMaxX</i> was encountered
2	a control character was encountered in the string

**WGREAD(2)****NAME**

**wgread - read window graphic input**

**SYNOPSIS**

```
#include <wuser.h>
wgread (fildes, buf)
struct Wgrec * buf;
```

**DESCRIPTION**

This function obtains a graphic input record for the active window specified. The conditions under which input records are generated are controlled by *ioctl(2)* WIOCENIN requests as defined in *window(4)*.

If the input mode is either IMNOINT or IMSAMPLE, the *wgread* call returns immediately in all circumstances. In any other mode, the calling process may be suspended until some graphic input condition is true.

*Parameters*

fildes /\*file descriptor of an open window file\*/

struct Wgrec {

short	grWTabX;	/* tablet, window relative*/
short	grWTabY;	/* tablet, screen relative*/
short	grSTabX;	/* real tablet coordinates*/
short	grSTabY;	/* (device dependent)*/
short	grTTabX;	/* logical button state*/
short	grTTabY;	/* number of graphic records still
char	grButtons;	queued after <i>wgread</i> call*/
char	grQsize;	/* timestamp, relative (60Hz)*/
char	grSpare;	
long	grTime;	

};

The physical puck buttons are mapped onto the logical buttons as specified in *wprofile(5)*.

Three standard logical buttons are defined by bit-significant constants:

WB_A	Set if button down
WB_B	Set if button down
WB_C	Set if button down

An additional logical button is provided for use with the optional four button puck and tablet:

WB\_DSet if button down

Additional pseudo-buttons are defined to report specific conditions:

WB_KEYBOARD	Set if keystrokes are available to the standard read interface (that is a <i>read(2)</i> would return immediately). Reported once, until either a <i>read</i> is executed or the input mode is reset
-------------	--

WB_DESEL	Set if the window was not selected at the time the record was generated. All other fields of the record are correct but should not normally be used when this bit is set
----------	--

## WGREAD(2)

Note: The button values `WB_YELLOW`, `WB_WHITE`, `WB_BLUE` and `WB_GREEN` are still defined for backward compatibility. They are not physical buttons but correspond to logical buttons A, B, C, D respectively. You must ensure the correct physical to logical mapping using `wprofile(5)`. The default setting maps yellow onto A, white onto B and both green and blue onto C.

### SEE ALSO

`ioctl(2)`, `window(4)`, `tablet(4)`, `wprofile(5)`.

### DIAGNOSTICS

<i>Value</i>	<i>Meaning</i>
0	successful
-1	fildes does not refer to an open window file (or, see also <code>errno</code> under <code>intro(2)</code> )

**WLINE(2)****NAME**

wline - draw a line on a window.

**SYNOPSIS**

```
#include <wuser.h>
wline (fildes, linargp, clipp)
struct LINCtl *linargp;
struct ClipCtl*clipp;
```

**DESCRIPTION**

The function draws a *line* between two end-points, specified relative to the window given by *fildes*. The line is drawn 1 pixel wide, and the logical operation used to draw it is specified. If the line has any points which would lie outside the window, or outside the specified clip area, then these points are ignored.

*Parameters*

*fildes* /\*file descriptor of an open window file\*/

struct LINCtl {

int	*LINDstBase;	/* Base of dest area (0 = window) */
short	LINX1 ;	/* 1st endpoint X displacement */
short	LINY1 ;	/* 1st endpoint Y displacement */
short	LINX2 ;	/* 2nd endpoint X displacement */
short	LINY2 ;	/* 2nd endpoint Y displacement */
short	LINStyle ;	/* line style */
short	LINDstInc ;	/* scan line increment */

};

struct *ClipCtl* is defined in *wrasop(2)*. An address of zero denotes no additional clipping

*Notes*

1. The line is drawn from (*LINX1,LINY1*) to (*LINX2,LINY2*), the latter point is not itself drawn.
2. If *LINDstBase* is not zero, it specifies an address in process store; only in this case is *LINDstInc* relevant (see *wrasop(2)*).
3. *LINStyle* has the following values, defined by constants:

<i>LDraw</i>	Draw line in foreground colour
<i>LErase</i>	Draw line in background colour
<i>LXor</i>	Draw line, inverting current pixel colour

**SEE ALSO**

*wrasop(2)*, *wdbyte(2)*, *rectangle(3)*, *window(4)*, *screen(4)*.

**DIAGNOSTICS**

<i>Value</i>	<i>Meaning</i>
0	successful
-1	<i>fildes</i> does not refer to an open window file (or, see also <i>errno</i> under <i>intro(2)</i> )



**WRASOP(2)****NAME**

wrasop - window raster operation

**SYNOPSIS**

```
#include <wuser.h>
wrasop (fildes, ropargp, clipp);
struct ROPCtl *ropargp;
struct ClipCtl*clipp;
```

**DESCRIPTION**

This function combines the values of the specified source and destination rectangles under the control of the specified operation. The source and destination areas can be either within the window specified, or user process data. If destination is the window, and the parameters specify that all or part of the area lies outside the window boundaries, then the operation is clipped. Further, the user may specify additional clipping to be applied to the destination. If the destination is the window, then whether the outcome is immediately visible or not depends on the actual screen layout; obscured sections are stored by the Window Management System and will appear if the screen layout is changed appropriately.

*Parameters*

fildes /\*file descriptor of an open window file\*/

struct ROPCtl {

int *ROPDstBase ;	/* base of dest area (0 = window) */
int *ROPSrcBase ;	/* base of source area (0 = window) */
short ROPHeight ;	/* height of area to rop */
short ROPWidth ;	/* width of area to rop */
short ROPDstInc ;	/* destination line inc */
short ROPSrcInc ;	/* source line inc */
short ROPSrcX ;	/* source X displacement */
short ROPSrcY ;	/* source Y displacement */
short ROPDstX ;	/* destination X displacement */
short ROPDstY ;	/* destination Y displacement */
short ROPFunc ;	/* RASTEROP function */

};

struct ClipCtl {

short ClipX; /* left clip boundary */	*/
short ClipY; /* top clip boundary */	*/
short ClipWidth; /* width of clip area */	*/
short ClipHeight; /* height of clip area */	*/

};

*Notes*

1. If either *ROPDstBase* or *ROPSrcBase* is non-zero, it specifies an address in process store which must be quad word aligned. Then the corresponding *Inc* parameter is relevant, and specifies the number of 16-bit units which comprise a *rasterline* in the area (must be a multiple of 4).

Areas addressed in process store (*LINDstBase*, *DBScreen*, *DBFont*) must conform to restrictions imposed by the hardware or the PNX kernel:

- (a) areas must be quad word aligned
- (b) areas must not cross page boundaries

The *rectangle(3)* library procedure supplies process store conforming to these restrictions

**WRASOP(2)**

2. *ROPFunc* specifies how the source and destination areas are to be combined. The following constants are defined:

<i>RRpl</i>	D := S
<i>RNot</i>	D := NOT S
<i>RAnd</i>	D := D AND S
<i>RAndNot</i>	D := D AND NOT S
<i>ROr</i>	D := D OR S
<i>ROrNot</i>	D := D OR NOT S
<i>RXor</i>	D := D XOR S
<i>RXNor</i>	D := D XNOR S

If the *clipp* parameter is not zero, then the destination area is clipped; if clipping occurs (by this or by window boundaries) the source area is adjusted by the same relative amounts.

**SEE ALSO**

*wline(2), wdbyte(2), rectangle(3), window(4), screen(4).*

**DIAGNOSTICS**

<i>Value</i>	<i>Meaning</i>
0	successful
-1	<i>fildes</i> does not refer to an open window (or, see also <i>errno</i> under <i>intro(2)</i> )

**BUILDTEXT(3)****NAME**

`buildtext` - builds an array of strings from a menu file

**SYNOPSIS**

```
buildtext (fname, menu);
char *fname;
struct menu_ctl *menu;
```

**DESCRIPTION**

`buildtext` reads the file *fname* and builds the structure *text* in the structure *menu*. The format of file *fname* is described in *menufile(5)*.

**FILES**

*menufile(5)*

**SEE ALSO**

*pickmenu(3)*

**DIAGNOSTICS**

<i>Returned value</i>	<i>Meaning</i>
0	Successful
-1	Could not open menufile
-2	Syntax error on menu file
-3	Insufficient memory



**CTYPE(3C)****NAME**

*isalpha*, *isupper*, *islower*, *isdigit*, *isxdigit*, *isalnum*, *isspace*, *ispunct*, *isprint*, *isgraph*, *iscntrl*, *isascii* - character classification

**SYNOPSIS**

```
# include <ctype.h>
int isalpha (c)
int c;
...
```

**DESCRIPTION**

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value EOF (see *stdio(3S)*).

<i>isalpha</i>	<i>c</i> is a letter
<i>isupper</i>	<i>c</i> is an upper case letter
<i>islower</i>	<i>c</i> is a lower case letter
<i>isdigit</i>	<i>c</i> is a digit [0-9]
<i>isxdigit</i>	<i>c</i> is a hexadecimal digit [0-9], [A-F] or [a-f]
<i>isalnum</i>	<i>c</i> is an alphanumeric
<i>isspace</i>	<i>c</i> is a space, tab, carriage return, new-line, vertical tab, or form-feed
<i>ispunct</i>	<i>c</i> is a punctuation character (neither control nor alphanumeric)
<i>isprint</i>	<i>c</i> is a printing character, code 040 (space) through 0176 (tilde)
<i>isgraph</i>	<i>c</i> is a printing character, like <i>isprint</i> except false for space
<i>iscntrl</i>	<i>c</i> is a delete character (0-177) or ordinary control character (less than 040)
<i>isascii</i>	<i>c</i> is an ASCII character, code less than 0200

**SEE ALSO**

*ascii(7)*



**GETOPT(3)****NAME**

*getopt* - get option letter from argv

**SYNOPSIS**

```
int getopt(argc, argv, opstring)
int argc;
char **argv;
char *opstring;
extern char *optarg;
extern int optind;
```

**DESCRIPTION**

*getopt* returns the next option letter in *argv* that matches a letter in *opstring*. *opstring* is a string of recognized option letters; if a letter is followed by a colon, the option is expected to have an argument that may or may not be separated from it by white space. *Optarg* is set to point to the start of the option argument on return from *getopt*.

*getopt* places in *optind* the *argv* index of the next argument to be processed. Because *optind* is external, it is normally initialized to zero automatically before the first call to *getopt*.

When all options have been processed (that is, up to the first non-option argument), *getopt* returns EOF. The special option — may be used to delimit the end of the options; EOF will be returned, and — will be skipped.

**EXAMPLE**

The following code fragment shows how you might process the arguments for a command that can take the mutually exclusive options a and b, and the options f and o, both of which require arguments:

```
main (argc, argv)
int argc;
char **argv;
{
    int c;
    extern int optind;
    extern char *optarg;
    .

    .

    while ((c = getopt (argc, argv, "abf:o:")) != EOF)
        switch (c) {
            case 'a':
                if (bflg)
                    errflg++;
                else
                    aflg++;
                break;
            case 'b':
                if (aflg)
                    errflg++;
                else
                    bproc();
                break;
            case 'f':
                ifile = optarg;
                break;
        }
}
```

**DIAGNOSTICS**

*getopt* prints an error message on *stderr* and returns a question mark (?) when it encounters an option letter not included in *opstring*.

**GETOPT(3)**

```
case 'o':  
    ofile = optarg;  
    bufsiza = 512;  
    break;  
case '?':  
    errflg ++;  
}  
if (errflg) {  
    sprintf (stderr, "usage: . . .");  
    exit (2);  
}  
for( ; optind < argc; optind++) {  
    if (access (argv[optind], 4)) {  
        . . .  
    }
```

**DIAGNOSTICS**

*getopt* prints an error message on stderr and returns a question mark (?) when it encounters an option letter not included in *optstring*.

**LPLOTA(3)****NAME**

*lplota* - copy window to area of memory

**SYNOPSIS**

```
#include, <plot.h>
lplota (file, control)
char file[];
struct plottable control;
```

**DESCRIPTION**

*lplota* copies the window given in *file* to a four-word-aligned area of memory. The location of this memory and the X and Y dimensions of the window image are passed back in the area pointed to by *control*.

Zero is returned by *lplota* unless an error is encountered in opening the file or in acquiring memory in which to put the window image, in which case -1 is returned and a message is written to the standard error file.

If the X dimension of the window is greater than 1024 then the window image is clipped for X = 1024.

If the window image is then too large for the area of memory its Y dimension is clipped accordingly.

This routine is kept in */lib/libu.a*.

**SEE ALSO**

*lplotb(3)*, *plot(1)*.



**L PLOTB(3)****NAME**

*lplotb* - copy window image to file

**SYNOPSIS**

```
#include <plot.h>
lplotb (file, control)
int file;
struct plottable control;
```

**DESCRIPTION**

*lplotb* writes the contents of an area of memory to the file whose file descriptor is given in *file*. It does not close the file. The area of memory and the area pointed to by *control* are assumed to have the form of the output from the library routine *lplota*. The format of the latter area is given in *plot.h* and includes a pointer to the image (the area to be written to the file) and its X and Y dimensions.

The first block of the file output is control information of format again to be found in *plot.h*. Most notably, the first four characters of this output are "PLOT".

Zero is returned unless an error is encountered, in which case -1 is returned and a message is written to the standard error file.

This routine is kept in */lib/libu.a*.

**SEE ALSO**

*lplot(3), plot(1), lpr(3)*.



**LPQ(3)****NAME**

*lpq* - generate unique filename  
*lpdq* - find oldest file in directory

**SYNOPSIS**

```
lpq_(file.pathname)
char*file.pathname

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/dir.h>
lpdq (directory, file, prefix)
char *directory, file, prefix
```

**DESCRIPTION**

*lpq* replaces the *file.pathname* template by a unique filename. The *file.pathname* should look like a file name ending in @@XXXXXX, the last 7 characters of which will be replaced by a unique letter and the current process id. The initial @ character is used during subsequent calls of *lpq* with a *file.pathname* which has already been returned by a previous call of *lpq*.

If a suitable filename is not available *lpq* returns a negative value.

This routine is kept in */lib/util.a*.

*lpdq* returns the name of the file, beginning with *prefix*, which was least recently put in the directory specified. *directory*, *filename* and *prefix* are pointers to null-terminated strings. *Prefix* may be the null string in which case *lpdq* returns the name of the oldest file in the directory.

*lpdq* returns -1 if no file with the given prefix exists in the directory.

This routine is kept in */lib/libu.a*.



**MENUESC(3)****NAME**

`menuesc` - routine to force escape from `pickmenu`

**SYNOPSIS**

`menuesc()`

**DESCRIPTION**

This routine may be called from `signal(2)` to force `pickmenu` to return to the calling process.

**SEE ALSO**

`pickmenu(3)`, `buildtext(3)`.



**PICKMENU(3)****NAME**

*pickmenu* - pop up menu package

**SYNOPSIS**

```
#include <mencntrl.h>
#include <wuser.h>
pickmenu (menu, frame)
struct Menu_ctl *menu
struct ClipCtl *frame
```

**DESCRIPTION**

*pickmenu* displays a pop up menu of commands in a specified open window then waits for a terminating event.

When *pickmenu* is invoked the tablet input becomes selected. The process samples puck button change of state or significant puck movement. *pickmenu* exits when:

- 1 A command is selected by pointing the cursor at a command and pressing and releasing puck button A
- 2 Button A is pressed and released when the puck cursor is outside the menu area
- 3 Q is typed at the keyboard
- 4 The process forces an exit by calling *menuesc(3)*

Parameters to *pickmenu* determine the appearance of the menu.

*Parameters*

*menu* is a pointer to a structure *MenuCtl*. This structure contains the following information:

```
struct menu-ctl {
    int w_desc;           /* file descriptor of an open
                           window file */ */
    struct text {
        char *title;      /* optional title. 0 = no title */ */
        unsigned *comnd_num; /* command numbers */ */
        char **cmnd_txt;   /* null terminated list of commands */ */
    };
    struct font *menu_font; /* address of font, 0 = default */ */
    unsigned *menu_slist;  /* pointer to variable into which
                           command selected is placed */ */
    short menu_posn;      /* specifies how and where menu
                           is to appear */ */
    short menu_svrst;     /* specifies whether menu area is to
                           be saved and restored */ */
};
```

*frame* is a pointer to a structure specifying the size of the menu:

```
struct ClipCtl {
    short ClipX;        /* menu left side x coordinate */ */
    short ClipY;        /* menu top side y coordinate */ */
    short ClipWidth;    /* width of menu */ */
    short ClipHeight;   /* height of menu */ */
};
```

## PICKMENU(3)

### NOTES

- 1 menu\_posn = \_WINDFIT /\* to fit menu to window \*/  
  |\_OPFIT /\* user to specify position \*/  
  |\_FRMFIT /\* position and size specified by frame \*/
- 2 menusvrst = |\_SAVE /\* to save and restore menu area \*/  
  |\_OVER /\* to overwrite window \*/
- 3 a macro *fileno(ptr)* is supplied to convert a window file pointer into a window descriptor

If *menu\_posn* is *\_OPFIT* or *\_FRMFIT* then frame must be supplied. If *menu\_posn* is *\_FRMFIT* then the menu position is specified by *frame*. If *menu\_posn* is *\_OPFIT*, position is determined by the user. *pickmenu* waits for a press on button A then displays the menu so that the top left corner appears at the position of the cursor. Note that *pickmenu* does not provide a message to remind users to press button A so the program calling *pickmenu* should. Also, the menu is clipped to the window so it should not be positioned too close to the edge.

### SEE ALSO

*buildtext(3)*, *menuesc(3)*.

### DIAGNOSTICS

<i>Returned value</i>	<i>Meaning</i>
0 or 1	Number of commands selected
-1	Parameter error
-2	Insufficient memory
-3	File descriptor does not refer to an open window or parameter error

## RECTANGLE(3)

### NAME

`rectangle` - main memory allocator for store to be used in graphics operations

### SYNOPSIS

```
char *rectangle(size)
unsigned size;
```

### DESCRIPTION

*rectangle* provides the means of allocating a store area, at run time, for use in graphics operations. A pointer to a block of store of at least *size* bytes is returned; *size* must be less than 128Kbytes, and the block will not cross a page boundary.

*rectangle* allocates the first suitable reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls *sbrk* (see *break*(2)) to get more memory from the system when there is no suitable space already free.

Pointers returned by *rectangle* may be used as parameters to *free* or *realloc* (see *malloc*(3)). In the case of *realloc*, the returned value is not guaranteed to be valid for calls to *wrasop*(2).

*rectangle* must be used instead of *malloc*(3) when allocating store for:

- 1 Font area for *wdbyte*(2)
- 2 Source areas for *wrasop*(2) it is not the window
- 3 Destination areas for *wdbyte*(2), *wline*(2) and *wrasop*(2) when not outputting to a window

### SEE ALSO

*wdbyte*(2), *wline*(2), *wrasop*(2), *malloc*(3).



**REGEX(3X)****NAME**

`regex, regcmp` - regular expression compile/execute

**SYNOPSIS**

```
char *regcmp(string1[,string2,...],0;
char *string1, *string2, ...;
char *regex(re,subject[,ret0, ...]);
char *re, *subject, *ret0, ...;
```

**DESCRIPTION**

`regcmp` compiles a regular expression and returns a pointer to the compiled form. `malloc(3C)` is used to create space for the vector. It is the user's responsibility to free un-needed space so allocated. A zero return from `regcmp` indicates an incorrect argument. `regcmp(1)` has been written to generally preclude the need for this routine at execution time.

`regex` executes a compiled pattern against the subject string. Additional arguments are passed to receive values back. `regex` returns zero on failure or a pointer to the next unmatched character on success. A global character pointer `lcl` points to where the match began. `regcmp` and `regex` were mostly borrowed from the editor, `ed(1)`. However, the syntax and semantics have been changed slightly. The following are the valid symbols and their associated meanings:

<code>[]*.</code>	These symbols retain their current meaning
<code>\$</code>	Matches the end of the string, <code>\n</code> matches the new line
<code>-</code>	Within brackets the minus means <i>through</i> . For example, <code>[a-z]</code> is equivalent to <code>[abcd...xyz]</code> . The <code>-</code> can appear as itself only if used as the last or first character. For example, the character class expression <code>[]-</code> matches the characters <code>]</code> and <code>-</code>
<code>+</code>	A regular expression followed by <code>+</code> means <i>one or more times</i> . For example, <code>[0-9]+</code> is equivalent to <code>[0-9][0-9]*</code>
<code>{m}</code> <code>{m,n}</code> <code>{m,u}</code>	Integer values enclosed in <code>{ }</code> indicate the number of times the preceding regular expression is to be applied. <code>m</code> is the minimum number and <code>u</code> is a number, less than 256, which is the maximum. If only <code>m</code> is present, <code>{m}</code> , it indicates the exact number of times the regular expression is to be applied. <code>{m,n}</code> is analogous to <code>{m,infinity}</code> . The plus (+) and star (*) operations are equivalent to <code>{1}</code> and <code>{0,}</code> respectively
<code>(... )\$n</code>	The value of the enclosed regular expression is to be returned. The value will be stored in the <code>(n+1)</code> th argument following the subject argument. At present, a maximum of ten enclosed regular expressions are allowed. <code>regex</code> makes its assignments unconditionally
<code>(... )</code>	Parentheses are used for grouping. An operator, for example <code>*, +, { }</code> , can work on a single character or a regular expression enclosed in parenthesis. For example, <code>(a*(cb+)*)\$0</code>

By necessity, all the above defined symbols are special. They must, therefore, be escaped to be used as themselves.

## REGEX(3X)

### EXAMPLES

Example 1:

```
char *cursor, *newcursor, *ptr;
...
newcursor = regex((ptr=regcmp("^\\n",0)),cursor);
free(ptr);
```

This example will match a leading new-line in the subject string pointed at by cursor.

Example 2:

```
char ret0[9];
char *newcursor, *name;
...
name = regcmp("[A-Za-z][A-za-z0-9]{0,7}",0);
newcursor = regex(name,"123Testing321",ret0);
```

This example will match through the string "Testing3" and will return the address of the character after the last matched character (cursor+11). The string "Testing3" will be copied to the character array *ret0*.

Example 3:

```
#include "file.i"
char *string, *newcursor;
...
newcursor = regex(name,string);
```

This example applies a precompiled regular expression in *file.i* (see *regcmp(1)*) against *string*

This routine is kept in */lib/libPW.a*.

### SEE ALSO

*ed(1)*, *regcmp(1)*, *malloc(3C)*.

### BUGS

The user program may run out of memory if *regcmp* is called iteratively without freeing the vectors no longer required. The following user-supplied replacement for *malloc(3C)* re-uses the same vector saving time and space:

```
/* user's program */
...
malloc(n) {
    static int rebuf[256];
    return&rebuf;
}
```

**STRING(3C)****NAME**

**strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strpbrk, strspn, strcspn, strtok - string operations**

**SYNOPSIS**

```
char *strcat (s1, s2)
char *s1, *s2;

char *strncat (s1, s2, n)
char *s1, *s2;
int n;

int strcmp (s1, s2)
char *s1, *s2;

int strncmp (s1, s2, n)
char *s1, *s2;
int n;

char *strcpy (s1, s2)
char *s1, *s2;

char *strncpy (s1, s2, n)
char *s1, *s2;
int n;

int strlen (s)
char *s;

char *strchr (s, c)          /* index in UNIX Version 7 */
char *s, c;

char *strrchr (s, c)         /* rindex in UNIX Version 7 */
char *s, c;

char *strpbrk (s1, s2)
char *s1, *s2;

int strspn (s1, s2)
char *s1, *s2;

int strcspn (s1, s2)
char *s1, *s2;

char *strtok (s1, s2)
char *s1, *s2;
```

**DESCRIPTION**

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

*Strcat* appends a copy of string *s2* to the end of string *s1*. *Strncat* copies at most *n* characters. Both return a pointer to the null-terminated result.

*Strcmp* compares its arguments and returns an integer greater than, equal to, or less than 0, according to whether *s1* is longer than, equal to or less than *s2*. *Strncmp* makes the same comparison but looks at, at most, *n* characters.

*Strcpy* copies string *s2* to *s1*, stopping after the null character has been moved. *Strncpy* copies exactly *n* characters, truncating or null-padding *s2*. The target may not be null-terminated if the length of *s2* is *n* or more. Both return *s1*.

*Strlen* returns the number of non-null characters in *s*.

### STRING(3C)

*Strchr (strrchr)* returns a pointer to the first (last) occurrence of character *c* in string *s*, or **NULL** if *c* does not occur in the string. The null character terminating a string is considered to be part of the string.

*Strpbrk* returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or **NULL** if no character from *s2* exists in *s1*.

*Strspn (strcspn)* returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

*Strtok* considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token, and will have written a **NULL** character into *s1* immediately following the returned token. Subsequent calls with zero for the first argument, will work through the string *s1* in this way until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, a **NULL** is returned.

### BUGS

*Strcmp* uses native character comparison, which is signed on PDP-11s, unsigned on other machines.

All string movement is performed character by character starting at the left. Thus overlapping moves toward the left will work as expected, but overlapping moves to the right may yield surprises.

**CG(4)****NAME**

cg - ICL 6202/02 Correspondence Printer

**DESCRIPTION**

The file `/dev/cg` gives access to the ICL 6202 Correspondence Printer when it is connected on the GPIB. The printer should be connected to the GPIB address 4 and should be declared as follows:

`/etc/mknod /dev/cg c 6 36`

The printer should be set as follows:

External switches enabled

Normal print at 6 lpi

Characters per inch determined by PITCH setting

11 inch paper (A4)

Tabs and margins cleared

**FILES**

`/dev/cg`

**SEE ALSO**

*gpiib(4)*



**CR(4)****NAME**

**cr - ICL 6202/03 Correspondence Printer**

**DESCRIPTION**

The file **/dev/cr** gives access to the ICL 6202 Correspondence Printer when it is connected to the RS232C interface. **/dev/cr** should be declared as follows:

**/etc/mknod /dev/cr c maj 2**

where **maj** should be **4** if connecting to the RS232C A port, or **5** if connecting to the RS232C B port.

The printer should be set up as follows:

External switches enabled

Normal print at 6lpi, 10cpi

9600 baud

**FILES**

**/dev/cr**

**/dev/rs**      Transparent RS232C printer interface. All device control to be performed by the user program

**SEE ALSO**

**srs(1), rs(4)**



**ERR(4)****NAME**

err - error-logging interface

**DESCRIPTION**

The system writes error records to */dev/err*, created by:

*/etc/mknod/dev/err c 3 0*

*/dev/err* may be opened by any process for writing and by one superuser process for reading. Each *write* adds a string error record to a queue of error records. Each read causes an entire error record to be retrieved; the record is truncated if the record request is for less than the record length.

**FILES**

*/dev/error*

**SEE ALSO**

*printerr(1M)*



**FLOP(4)****NAME**

flop - PERQ floppy disc

**DESCRIPTION**

PNX supports three different types of floppy using several different minor devices. The 3 types are:

- 1 Standard PNX filestore floppy disc (*/dev/flop*)
- 2 Old PNX filestore floppy disc, that is, previous to PNX release 2.0 (*/dev/oldflop*)
- 3 RT-11 format floppy disc, for example, POS filestore floppy disc (*/dev/fdplx*, */dev/flop*)

Boot floppy discs are a special case of types 1 and 2 in that the first five tracks are reserved for the bootstrap and they must be single density.

Standard filestore floppy discs have hardware interleaving and may be single or double density. Blocks consist of logically consecutive sectors.

Old PNX filestore floppies have software interleaving and may be single or double density.

RT-11 (POS) type floppy discs may be single or double density.

The file */dev/flop* may also be used for raw access to the device (replacing the old device */dev/rawflop*). The *open(2)* call determines whether the disc is single or double density. Blocks consist of logically consecutive sectors (4 in single density, 2 in double density) starting at sector 1, track 0.

**FILES**

The following table lists the floppy devices used by PNX together with their device description to */etc/mknod* and a description of their use:

<i>/dev/flop</i>	b 1 1	Standard PNX floppy disc, single or double density, single or double sided.  Single density = 26 128 byte sectors/track  = 6\ blocks/track. (Blocks are numbered 0-1000)
		Double density= 26 256 byte sectors/track = 13 blocks/track. (Blocks are numbered 0-2001)
<i>/dev/bflop</i>	b 1 0	Standard PNX boot floppy disc. Boot floppy discs must be single density. The first 5 tracks are reserved for the bootstrap. Only 24 sectors on each track are used for the filestore
<i>/dev/formatfd</i>	b 1 4	For formatting single or double density floppy discs. Used by the <i>f(1)</i> command
<i>/dev/fdplx</i>	b 1 5	RT-11 type floppy disc. Used by <i>dir</i> , <i>get</i> , <i>sget</i> and <i>put</i> commands under <i>f(1)</i>

**FLOP(4)**

/dev/oldflop	b 1 6	Old PNX file system floppy disc with software interleaving. Only 24 sectors on each track used
/dev/oldbflop	b 1 7	Old PNX boot floppy disc with software interleaving

**SEE ALSO**

*mkflop(1), mountflop(1).*

**GPIB(4)****NAME**

GPIB interface

**DESCRIPTION**

This section describes the GPIB interface.

To access devices on the GPIB, a special file corresponding to the device must first be created by issuing the command:

**etc/mknod /dev/name c 6 n**

where

*name* can be any identifier

*n* is the device address on the GPIB. (In PNX, 6 is the major device number of the GPIB.)

The GPIB is initialised by PNX when multiuser mode is invoked by the program, *gpsrq* that handles SRQ interrupts (see below). The initialisation consists of a reset to the TMS 9914 GPIB chip and an Interface Clear to the bus. Note: REN (Remote Enabled) is not set by this initialisation process.

If an error occurs on the GPIB which implies that the bus is locked up, for example, timeout on a write, this initialisation is performed again and may affect other users of the bus.

The device can then be accessed using the system calls *open(2)*, *close(2)*, *read(2)*, *write(2)*, and *ioctl(2)*.

All these system calls are implemented with the PERQ as the controller on the bus. With the current hardware the PERQ must be the system controller on the bus.

A special file set up with minor device number 31 can be used to read and write to the GPIB bus directly, in command or data mode as specified by the *ioctl(2)* interface.

A special file set up with minor device number 32 can be used to write to the TMS 9914 registers with data in the form:

<regno,databyte>\*

or read the sense data, for example after an attention interrupt when exclusive control is in force (see below). 10 bytes of data are returned as follows (contents of TMS 9914 registers)

(Saved) INSTATO INSTATI ADDRSTAT BUSSTAT ADDRSW1 CMDPASS  
(Current) ADDRSTAT BUSSTAT ADDRSW1 CMDPASS

Detailed knowledge of the Texas Instruments chip is required in order to use this facility.  
*open(2)* returns an identifier to be used in subsequent I/O calls but provokes no I/O.

*read(2)* causes the device to be set up as talker, and PERQ as listener (unless modified by *ioctl(2)* see below) and is terminated when the buffer is filled, EOI is detected, or a designated terminator is found.

*write(2)* causes the device to be set up as listener, and PERQ as talker, and the data is sent.

## GPIB(4)

If the system tablet is the optional high resolution tablet connected on the GPIB, the kernel may sample it between transfers (or in the middle of low-volume transfers of greater than 256 bytes). The kernel does not buffer data read from devices and discards data that arrives when a read is not in progress. Programs that do not read all the data that a device may send in one read should use the *ioctl* call GPIBLIM to limit the device (see below)

The precise actions of *read*(2) and *write*(2) can be modified by *ioctl*(2) calls as follows:

The following *ioctl*(2) calls for teletype files are applicable to GPIB special files. They have the form:

```
#include sgtty.h
ioctl (fildes, code, arg)
```

The *arg* is ignored. The applicable codes are:

**TIOCEXCL** Set the "exclusive use" mode (N.B. of the *file* not of the GPIB). No further opens of the file are permitted until the file has been closed

**TIONXCL** Turn off "exclusive use" mode

The following *ioctl*(2) codes are specific to the GPIB and have the general form:

```
# include sgppib.h
ioctl (fildes, code, arg)
```

In the following codes the argument is a pointer to the following structure, defined in *sgppib.h*:

```
struct gpibdata {
    char t_chars[2];      /* 2 possible read terminators */
    int r_timeout;        /* time out for reads in 1/60 sec */
    int w_timeout;        /* time out for writes in 1/60 sec */
    char sec_addr;        /* secondary address */
    char srq_status;      /* last gpib status received (from
                           serial poll) */
```

}

The default values for these fields are:

0,0 i.e. no terminators. If either field is set to non-zero then both are checked as possible terminators unless ESC (hex 1B) is set in the first byte. ESC in the first byte means only the second byte is to be checked as a terminator (so Esc, 0 means check for 0 terminator).

Note: This only applies to low volume transfers.

600 read time out of 10 seconds

600 write time out of 10 seconds

hex FF no secondary addressing. If this field is changed to anything else, it is used as a secondary address

*srq\_status* is an eight bit value reporting the device status. Bit value 2<sup>6</sup> on indicates that the device has a Service Request

The calls have the form:

```
#include sgppib.h
ioctl (fildes, code, arg)
struct gpibdata *arg;
```

The codes are

**GPIBGETD** Fetch the GPIB file data. Use this code to fetch the data in *gpibdata* before changing it and reading it back with GPIBSETD

**GPIBSETD** Set the GPIB file data

**GPIB(4)**

gpibstatus, an integer giving further information after a read or write system call, is returned by the following call:

```
#include (sggpib.h)
ioctl (fildes, code, arg)
struct gpibstatus *arg;
struct gpibstatus {
    int      result; /* bit significant I/O result field */
};
```

The code is

**GPIBSTATUS**

Provide status information following an I/O operation. The possible status settings are defined in /sys/sggpib.h. They are:

```
/* define low level bit settings */
GPIBERROR      /* an error has occurred */
GPIBTIMEOUT    /* timeout on transfer */
GPIBDL         /* data overrun on input */
GPIBTFND       /* specified terminator found */
GPIBFB          /* buffer filled on read */
GPIBEFND       /* EOI found on read */
GPIBNAK         /* NAK received (illegal write to registers)*/
GPIBSERR        /* device not present on write */
```

All the following *ioctl* codes ignore the argument field

<b>GPIBEXCL</b>	Acquire exclusive control of GPIB synchronously
<b>GPIBNXCL</b>	Release exclusive control immediately
<b>GPIBIXCL</b>	Ignore exclusive control of GPIB (used if one program opens files on more than one device and acquires exclusive control on one of them)
<b>GPIBC</b>	All subsequent writes are performed in command mode, that is, with attention set (mainly for minor device 31)
<b>GPIBD</b>	All subsequent writes are performed in data mode, that is with attention in unset (this is the default)

The following codes modify the action of *read(2)* and *write(2)* system calls as described. The effect of any of these calls is cancelled by repeating the *ioctl* call

<b>GPIBEOIR</b>	Terminate read on receipt of EOI (high volume reads always terminate if EOI is detected)
<b>GPIBEOIW</b>	Send EOI on last byte of write (low volume only)
<b>GPIBNOCONF</b>	Do not untalk, unlisten, address device, and so on before each transfer
<b>GPIBNOUNLISTEN</b>	Do not send unlisten when addressing device. This enables a device to be set up as a listener, (for example, a printer), before reading from another device. The first device then also receives the data read

**GPIB(4)**

<b>GPIBHIVOL</b>	Perform transfers in high volume mode direct from user's buffer. The buffer must be 4 word aligned and should therefore be acquired by calling <i>rectangle</i> (3). For PERQ2, the buffer must also be a multiple of 8 bytes long though the length specified by <i>write</i> can be any length greater than 1. When reading random data is included beyond the end of the transfer to fill the space up to the 8 byte aligned boundary.
	The default state is low volume mode. High volume transfers impose less of an overhead on the kernel but some <i>ioctl</i> facilities are unavailable for high volume transfers.
<b>GPIBSYNC</b>	Return from write immediately, before write has terminated. A second call with this code unsets this feature. GPIBSYNC must be unset before the last write. Errors can only be reported on the last write. This feature is intended for programs that have to keep up with fast devices that need a large amount of data.
	GPIBSYNC can only be used if GPIBHIVOL is in force and only applies to writes.

**GPIBLIM**      Read message using 'hold off on all data', that is, data is sent byte by byte to a user's buffer. NRFD (Not Ready for Data) is not set to false until a byte has been read into the buffer. In high volume reads the last byte is read in this way.

This avoids possible data loss between successive reads if other devices are being accessed on the bus

The following commands are for devices that may generate SRQ:

**GPIBSRQ**      Inform PNX that device can generate SRQ and that the device should be added to the queue of devices to be serially polled on SRQ interrupts. A maximum of eight devices may be added to the serial poll queue

**GPIBENDSRQ**    SRQ processing complete. SRQ handling is further described below

The following codes cause an I/O to be performed to the device:

**GPIBCLEAR**     Can have one of three effects depending on the device:

SDC (selected device clear) on minor device 0 to 30

DCL (device clear) on minor device 31

IFC (interface clear) on minor device 32

**GPIBGET**       Trigger (Group Execute Trigger)

**GPIBREN**       Set REN (Remote Enable)

**GPIBCLO**       Unset REN that is, clear lockout

**GPIBLOCAL**     Issue GTL (Go to local)

**GPIBLLO**       Set local lockout

**GPIBTALK**      Set device as a talker

**GPIBLISTEN**    Set device as a listener

If an SRQ interrupt occurs when exclusive control is not in force, all devices for which a GPIBSRQ *ioctl*(2) has been issued are serially polled. A *signal*(2) SIGGPIB is sent to every program with a file open for a device that is indicating SRQ.

The program should issue an *ioctl*(2) GPIBGETD to read the status for all its open files until it finds one with a status indicating SRQ.

**GPIB(4)**

That program should then perform any I/O that is appropriate, then indicate that it has finished processing the interrupt by issuing a GPIBENDSRQ *ioctl(2)*. This causes the status field *srq\_status* in *gpibdata* to be cleared to zero and the next file with SRQ set can be signalled.

If exclusive control is in force when any attention interrupt (which may not have been caused by SRQ) is received by the PERQ and the program has indicated its willingness to handle them by issuing a GPIBSRQ *ioctl(2)*, then after the GPIBEXCL *ioctl* call the *signal(2)* SIGGPIB is sent to the process. It would then normally issue a read of 10 bytes on a file opened on minor device 32 to fetch the sense data. This facility is intended for specialised programs which wish to make full use of the facilities of the TMS 9914 chip.

However if exclusive control is in force on a file which has not issued a GPIBSRQ *ioctl(2)*, then the servicing of the SRQ is held up until exclusive control is released.

**FILES**

/dev/tablet  
any GPIB special file

**SEE ALSO**

*ioctl(2)*, *tty(4)*, *signal(2)*, *rectangle* (3).



**HARD(4)****NAME**

hard - PERQ fixed disc

**DESCRIPTION**

The file *hard* gives access to the PERQ fixed disc as a single sequentially-addressed file. Its 512 byte blocks are numbered from 0. The accessible data includes the process swap area and the UNIX filestore but excludes the bootstrap area.

The disc is accessed using direct transfers between the disc and the user's read or write buffer if possible, otherwise using the system's normal buffering mechanism. In the direct transfer case, the buffer must begin on a four-word boundary and counts should be a multiple of 512 bytes. Likewise, *lseek* calls should specify a multiple of 512 bytes. (Direct transfer is not implemented in this version of PNX.)

**FILES**

/dev/hard	Normal hard disc device, logical block number 0 is at physical block 300
/dev/veryhard	Used only for writing bootstraps. Logical block number 0 is at physical block 0



**LP(4)****NAME**

*lp* - ICL 3185 Matrix Printer

**DESCRIPTION**

The file *lp* gives access to the ICL 3185 Matrix Printer when it is connected to the RS232C interface. The file should be declared as follows:

**etc/mknod /dev/lp C maj 1**

where *maj* is **4** if connecting to the RS232C A port and **5** if connecting to the RS232C B port.

**FILES**

**/dev/lp** ICL 3185 Matrix Printer interface. It uses the RS232C interface as follows:

- \* 8-bit characters
  - \* No parity
  - \* All other values set up by SRS(1)
- It uses the printer as follows:
- \* Ordinary character generator
  - \* Horizontal tabs every 8 character positions
  - \* Rotary switch enabled for form length
  - \* 6 lines per inch
  - \* 10 characters per inch

**/dev/rs** Transparent RS232C printer interface. All device control to be performed by the user program.

**SEE ALSO**

*srs(1), rs(4).*



RS(4)

**NAME**

rs,rsa rsb - general purpose RS232C interface driver

**DESCRIPTION**

This section describes the interface associated with the special files */dev/rs*, */dev/rsa* and */dev/rsb*. */dev/rsa* and */dev/rsb* identify the interface to devices or communications lines connected to the RS232C A and RS232C B ports. */dev/rs* and */dev/rsa* are equivalent.

The interface allows both input and output. Input may occur simultaneously with output; separate buffering space is used. Input characters are lost when the system's input buffers have accumulated the maximum allowed number of input characters that have not yet been read by some program. When the input limit is reached the incoming character replaces the last buffered character.

If this interface is used for file transfer at or above 9600 baud, it will be found necessary to specify a count of 128 bytes in the *read(2)* system call, and also in the associated *write(2)* system call, in order to avoid overrun of the system buffer with consequent loss of data.

Reads may be for any number of characters, including one. All characters are passed to the reading program.

If parity is specified, then the parity bit is stripped off on input.

A status error code may be reported on the screen as a result of a *read(2)* system call. The status is from the SIO controller chip's read register 1 and contains the following bit encoded information:

7	6	5	4	3	2	1	0
							+ All sent
							+ + + Residue data
							+ Parity error
							+ Receiver Overrun Error
							+ CRC/Framing error
							+ End of frame (SDLC)

Error codes of FF and -2 have special meanings unrelated to the bit map above. FF means that the last character just read arrived while the input buffer was full. Thus, some characters were lost. -2 means that the Z80 firmware lost some characters. Due to buffering of input characters, there no way to tell where in the input stream the characters were lost so recovery can be difficult.

Several *ioctl(2)* calls apply to the devices on the RS232C interface. They have the following form:

```
#include<sgtty.h>
ioctl(filedes, code, arg)
struct rs232b arg
```

*filedes* is returned by *open*. *Code* is one of; TIOCGETP, TIOCSETP, TIOCGETA, TIOCSETA, TIOCRSET. *Arg* points to the following structure defined in *<sgtty.h>*

```
struct rs232b {
```

```
char    rs_speed;
short   rs_flags;
char    rs_term;
char    rs_br;
char    rs_bt;
char    rs_st;
char    RecvSpeed;
char    XmitSpeed;
```

**RS(4)**

```

union {
    struct {
        char Read0;
        char Read1;
        char Write0;
        char Write3;
        char Write4;
        char Write5;
        char Write6;
        char Write7;
    } Reg;
    struct {
        unsigned EndOfFrame :1;
        unsigned CRCFramingError :1;
        unsigned RxOverrunError :1;
        unsigned ParityError :1;
        unsigned Residue :3;
        unsigned AllSent :1;
        unsigned BreakAbort :1;
        unsigned SendingCRCSync :1;
        unsigned CTS :1;
        unsigned SyncHunt :1;
        unsigned DCD :1;
        unsigned TxBufEmpty :1;
        unsigned IntPending :1;
        unsigned RxChrAvailable :1;
        unsigned RxBits :2;
        unsigned AutoEnables :1;
        unsigned EnterHuntMode :1;
        unsigned RxCRCEnable :1;
        unsigned AddrSrchMode :1;
        unsigned SyncChrLdInhib :1;
        unsigned RxEnable :1;
        unsigned ResetCRC :2;
        unsigned Command :3;
        unsigned NxtRegPtr :3;
        unsigned DTR :1;
        unsigned TxBits :2;
        unsigned SendBreak :1;
        unsigned TxEnable :1;
        unsigned SDLCCRC16 :1;
        unsigned RTS :1;
        unsigned TxCRCEnable :1;
        unsigned ClockRate :2;
        unsigned SyncMode :2;
        unsigned StopBits :2;
        unsigned Parity :2;
        unsigned SynChar1 :8;
        unsigned SynChar0 :8;
    } Bits;
} SIOChip;

```

**TIOCRSET** Reset the RS232 to the default state. Input and output characters may be lost. This code enables you to get the RS232 out of a hung state

**TIOCGETP** Fetch the parameters associated with the device set by TIOCSETP, and store in the pointed to structure. Only the fields rs\_speed, rs\_flag, rs\_term, rs\_br, rs\_bt and rs\_st are meaningful

**TIOCGETA** Fetch the parameters associated with the device set by TIOCSETA and store in the pointed to structure. The fields rs\_speed, rs\_br, rs\_bt and rs\_st have no meaning for this code

**RS(4)**

**TIOCSETP** Set the parameters according to the pointed to structure. In most cases the TIOCGETA command should be used. Only the codes meaningful for TIOCGETP are meaningful for this code:

*rs\_speed* - sets the input and output baud rates, must be

RExt	0	use external clocking (PERQ 2 only)
R100	1	100 baud
R150	2	150 baud
R300	3	300 baud
R600	4	600 baud
R1200	5	1200 baud
R2400	6	2400 baud
R4800	7	4800 baud
R9600	8	9600 baud
R19200	9	19200 baud (PERQ 2 only)

PERQ 1 requires input and output speeds to be the same.

*rs\_flags* - sets parity and other modes, defined bits are:

EVENP	0000200	set even parity on transmit and receive
ODDP	0000100	set odd parity on transmit and receive
ANYP	0000300	set any parity on receive, no parity on transmit. Setting this means the PERQ ignores any parity bits on characters sent
RAW	0000040	Do not use the termination character. Allow reads to terminate before reading if there are no characters to read but only if at least one character has been read
ENABLEIN	0004000	Enable the receiving of characters
QUIET	0001000	Disable reporting of errors to the screen.

*rs\_term* - sets the termination character. If this character is received, the PERQ treats it as if it received the end of file. The current read terminates and future reads will return a value of -1 and errno will be set to EOF. If rs is open with write, this character will be sent when the file is closed

*rs\_br* - sets the number of bits that make up a receive (*rs\_br*)\ transmit (*rs\_bt*) character, this includes the parity bit. *rs\_br* and *rs\_bt* must be one of:

No Parity		Parity set	
BIT5	0 5 data bits	4 data bits,	1 parity bit
BIT6	1 6 data bits	5 data bits,	1 parity bit
BIT7	2 7 data bits	6 data bits,	1 parity bit
BIT8	3 8 data bits	7 data bits,	1 parity bit

*rs\_st* - sets the number of stop bits at the end of each character, must be one of:

STOP1	0	1 stop bit
STOP2	1	1.5 stop bits
STOP3	2	2 stops bits

**RS(4)**

**TIOCSETA** Set the parameters according to the structure pointed to. Fields which do not have a meaning for TIOCGETA do not have a meaning for this code. Applicable fields are:

*rs\_flags* - sets the following modes, defined bits are:

QUIET	0001000	Disable reporting of errors to the screen. Do not use the termination character.
RAW	0000040	Allow reads to terminate before reading the number of characters specified if there are no characters to read but only if at least one character has been read

*rs\_term* -

RecvSpeed - sets the receive baud rate, otherwise identical to use of *rs\_speed* in TIOCSETP

XmitSpeed - sets the transmit baud rate, otherwise identical to use of *rs\_speed* in TIOCSETP

The union SIOChip.

The union SIOChip provides two ways to look at the rest of the structure. The first is as a collection of characters. The second is as a bit map of these characters

The Read0 and Read1 fields provide the value of the SIO chip's read register 0 and read register 1.

The Write0 field specifies the setting of the SIO chip's write register 0. The bits for this register are ResetCRC, Command, and NxtRegPtr.

The ResetCRC bits can have one of the following values:

NullCode	0	Null Code (no effect)
RxCRCChkReset	1	Reset receive CRC checker
TXCRCGenReset	2	Reset transmit CRC generator
SendingCRCReset	3	Reset Sending CRC/Sync latch

The command bits should have one of the following values:

NullCommand	0	No effect
SendAbort	1	Generate 8 to 13 ones (SDLC only)

The NxtRegPtr bits should not be changed.

The Write3 field specifies the SIO chip's write register 3. The bits for this register are RxBits, AutoEnables, EnterHuntMode, RxCRCEnable, AddrSrchMode, SyncChrLdInhib, and RxEnable.

RxBits specifies the number of bits that form an input character and contains one of:

Rx5Bits	0	5 bit characters
Rx6Bits	2	6 bit characters
Rx7Bits	1	7 bit characters
Rx8Bits	3	8 bit characters

For the following bits, 1 enables and 0 disables:

AutoEnables	Selects Auto Enable mode
EnterHuntMode	Reenters Hunt mode
RxCRCEnable	Enables receiver CRC
AddrSrchMode	Enables Address search mode
SyncChrLdInhib	Enables sync character load inhibit
RxEnable	Enables transmission of characters

**RS(4)**

The Write4 field specifies the SIO chip's write register 4. The bits for this register are ClockRate, SyncMode, StopBits, and Parity.

Parity specifies the parity of transmitted and received characters and contains one of:

NoParity	0	No parity bits
OddParity	1	Character sent/check has odd parity
EvenParity	3	Character sent/check has even parity

StopBits specifies the number of stop bits added to each character sent and contains one of:

SynModeEnable	0	Synchronous mode selected
Stop1Bit	1	1 stop bit
Stop1x5Bits	2	1.5 stop bits
Stop2Bits	3	2 stop bits

SyncMode specifies the character synchronisation option and contains one of:

Sync8bit	0	8-bit sync
Sync16bit	1	16-bit sync
SDLCMode	2	SDLC mode
SyncExt	3	External Sync Mode

ClockRate specifies the multiplier between the clock and data rates and contains one of:

X1	0	Data rate x 1 = clock rate
X16	1	Data rate x 16 = clock rate
X32	2	Data rate x 32 = clock rate
X64	3	Data rate x 64 = clock rate

The Write5 field specifies the SIO chip's write register 5. The bits for this register are DTR, Txbits, SendBreak, TxEnable, SDLCCRC16, RTS and TxCRCEnable.

TxBits specifies the number of bits that form an output character and contains one of:

Tx5Bits	0	Transmit 5 bits or less
Tx6Bits	2	Transmit 6 bits per character
Tx7Bits	1	Transmit 7 bits per character
Tx8Bits	3	Transmit 8 bits per character

For the following bits, 1 enables and 0 disables.

DTR	Activates the DTR pin
SendBreak	Activates the TxD pin
TxEnable	Enables transmission of characters
SDLCCRC16	Selects the CRC mode
RTS	Activates the RTS pin
TxCRCEnable	Determines if a CRC is to be calculated

The Write6 field specifies the first 8 bits of a BiSync sequence or the check address in SDLC mode.

The Write7 field specifies the second 8 bits of a BiSync sequence.

The include file, <sgtty.h> contains the following macros which allow easy access to Read0, Read1, Write0, and so on.

ReadReg0	for	SIOChip.Reg.Read0
ReadReg1	for	SIOChip.Reg.Read1
WrtReg0	for	SIOChip.Reg.Write0
WrtReg3	for	SIOChip.Reg.Write3
WrtReg4	for	SIOChip.Reg.Write4
WrtReg5	for	SIOChip.Reg.Write5
WrtReg6	for	SIOChip.Reg.Write6
WrtReg7	for	SIOChip.Reg.Write7

The line characteristics set by the *ioctl(2)* call remain set when the file is closed (they are not reset to the default characteristics).

**RS(4)**

The default state of the RS232 is:

rs_speed	-	R9600
rs_flags	-	ENABLEIN
rs_term	-	'C (ETX or hex 03)
rs_br	-	BIT8
rs_bt	-	BIT8
rs_st	-	STOP2
RecvSpeed	-	R9600
XmitSpeed	-	R9600
ReadReg0	-	Undefined
ReadReg1	-	Undefined
WrtReg0	-	0x40 (Rx CRC ChckReset, NullCommand)
WrtReg3	-	0xE1 (RxEnable, AutoEnables, Rx8Bits)
WrtReg4	-	0x48 (NoParity, Stop1x5Bits, Sync 8bit, x16)
WrtReg5	-	0xEA (DTR, Tx8bits, TxEnable, RTS)
WrtReg6	-	0
WrtReg7	-	0

No other copies have any effect.

**FILES**

/dev/rs  
/dev/rsa  
/dev/rsb

**SEE ALSO**

srs(1), signal(2), ioctl(2)

**RSTTY(4)****NAME**

*rstty* - general terminal interface via the RS232C interface

**DESCRIPTION**

This section describes the interface associated with the special file */dev/rstty*.

As for terminals in general: the same general interface is used as for low-speed asynchronous communications ports on standard PNX. The remainder of this section discusses the common features of the interface.

This file may be opened in three ways:

- 1 By the user's program
- 2 By *init(8)*
- 3 By *rstty(1)*

If the file is opened by *init(8)* or *rstty(1)* it will be available as a standard PNX *control terminal* in a separate process group. The file is opened by *init(8)* if an appropriate entry exists in the file */etc/ttys* (see *ttys(5)*).

Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely choked, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. When the input limit is reached characters may be lost.

Special characters (DEL, ^Z, ^C, and so on) are handled as specified in *tty(4)*.

The *ioctl(2)* interface (and associated structure *sgtty.h* and codings) is as specified in *tty(4)* with the following exceptions.

- 1 The applicable codes are:

TIOCGETP/TIOCGETA  
TIOCSETP/TIOCSETA

Each pair is treated identically.

The following fields in structure *sgttyb* (see *tty(4)*) affect the RS232 interface:

<i>sg_ispeed</i>	Determine the baud rate of the RS232C
<i>sg_ospeed</i>	Must be the same as <i>sg_ispeed</i> and must be one of B110, B150, B300, B600, B1200, B2400, B4800 or B9600
<i>sg_flags</i>	The EVENP, ODD and ANYP bits determine the parity of characters sent and received on the RS232C

Note: *rstty* assumes that terminals connected to it transmit and receive eight bits per character (7 bits data, 1 bit parity or 8 bits data, no parity). It is not possible to change this. It is possible to change the number of stop bits expected using *srs(1)*.

The protection, against interference between users at the PERQ console and RS232 control terminal should be set up using the standard PNX mechanism that is, login security, (including alternative shell) and file access permissions.

**SEE ALSO**

*stty(1)*, *rstty(1)*, *signal(2)*, *ioctl(2)*.



**SCREEN(4)****NAME**

/dev/screen - PERQ physical screen interface

**DESCRIPTION**

The file */dev/screen* is a special file which gives access to the PERQ screen for graphical operations. Use of */dev/screen* does not require the Window Management System to be running, but as it is supported by the Window Driver, this must be included in the kernel in use.

The file */dev/screen* is made by issuing the command:

*/etc/mknod /dev/screen c 8 0*

When this device is opened, the system calls *wdbyte(2)*, *wline(2)* and *wrasop(2)* may be used to write to the screen. Window relative co-ordinates are coincident with screen-relative co-ordinates.

The cursor mode is initially CMOFF. To enable the cursor, set the required mode. To connect the tablet driver to the cursor and to receive input from the tablet through *wgread* it is necessary to open */dev/tablet* (see *tablet(4)*). Tablet information is accessed through *wgread* calls not through calls to */dev/tablet*.

Window controls (*ioctl(2)*) as defined in *window(4)* also apply to */dev/screen* except for window update control, preserve/delete control and the omission of text information in window status.

Use of */dev/screen* while the Window Management System is running may interfere with the Window Management System which takes no account of its use. In general, it is safe to read screen areas; writing may corrupt displayed windows; neither graphic nor keyboard input is possible as the screen cannot be selected by the operator.

The screen device has a blank cursor pattern at system load; the Window Management System sets the screen cursor pattern to be the default selected pattern at Window Management System initialisation.

The screen colour implied by the screen device cursor function is used by the Window Management System to determine the basic screen colour for all windows. This can be changed at any time.

Text input and output via the keyboard and screen is supported by the driver defined in */dev/console*.

**FILES**

*/dev/screen*

**SEE ALSO**

*wdbyte(2)*, *wline(2)*, *wrasop(2)*, *wgread(2)*, *window(4)*.



**SPEECH(4)****NAME**

speech - general purpose speech driver

**DESCRIPTION**

Speech is a special file that allows PNX to produce sound. A *write(2)* to speech produces sound according to the bit pattern written.

Several *ioctl(2)* calls apply to speech. They should be used only by those familiar with the speech hardware. The *ioctl(2)* calls use the following union:

```
union speechb {
    struct {
        char      sp_Read0;
        char      sp_Read1;
        char      sp_XmitRate;
        char      sp_Write4;
        char      sp_Write5;
        char      sp_Write6; } sp_Regs;
    struct {
        unsigned sp_EndOfFrame : 1;
        unsigned sp_CRSFraming_Error : 1;
        unsigned sp_RxOverrunError : 1;
        unsigned sp_ParError : 1;
        unsigned sp_Residue : 3;
        unsigned sp_AllSent : 1;
        unsigned sp_BreakAbort : 1;
        unsigned sp_SendingCRCSync : 1;
        unsigned sp_CTS : 1;
        unsigned sp_SyncHunt : 1;
        unsigned sp_DCD : 1;
        unsigned sp_TxBufEmpty : 1;
        unsigned sp_IntPending : 1;
        unsigned sp_RxChrAvailable : 1;
        unsigned : 16;
        unsigned sp_DTR : 1;
        unsigned sp_TxBits : 2;
        unsigned sp_SendBreak : 1;
        unsigned sp_TxEnable : 1;
        unsigned sp_SDLCCRC16 : 1;
        unsigned sp_RTS : 1;
        unsigned sp_TxCRCEnable : 1;
        unsigned sp_ClockRate : 2;
        unsigned sp_SyncMode : 2;
        unsigned sp_StopBits : 2;
        unsigned sp_Parity : 2;
        unsigned : 8;
        unsigned sp_SynChar0 : 8;
    } sp_Bits;
}
```

The *Regs.Read0* field provides the value of the SIO chip's read register 0. The fields *Bits.BreakAbort .. Bits.RxChrAvailable* give a bit mapping of this field.

The *Regs.Read1* field provides the value of the SIO chip's read register 0. The fields *Bits.BreakAbort .. Bits.RxChrAvailable* give a bit mapping of this field.

The field *Regs.XmitRate* specifies the clock count to be loaded into the CTC port of the SIO chip. It has no bit mapping.

The field *Regs.Write4* specifies the SIO chip's write register 4. The fields *Bits.ClockRate .. Bits.Parity* give the bit mapping of this field.

**SPEECH(4)**

Bits.ClockRate contains one of:

X1	0	clock rate = 1 X data rate
X16	1	clock rate = 16 X data rate
X32	2	clock rate = 32 X data rate
X64	3	clock rate = 64 X data rate

Bits.SyncMode contains one of:

Sync8bit	0	8-bit sync
Sync16bit	1	16-bit sync
SDLCMode	2	SDLC mode
SyncExt	3	External Sync Mode

Bits.StopBits contains one of:

SynModeEnable	0	Synchronous mode selected
Stop1Bit	1	1 stop bit
Stop1x5Bits	2	1.5 stop bits
Stop2Bits	3	2 stop bits

Bits.Parity contains one of:

NoParity	0	No parity bits
OddParity	1	Characters sent have odd parity
EvenParity	2	Characters sent have even parity

The field Regs.Write5 specifies the SIO chip's write register 5. The fields Bits.DTR ... Bits.TxCRCEnable give the bit mapping of this field.

Bits.TxBits contains one of:

Tx5Bits	0	Transmit 5 bits or less
Tx6Bits	2	Transmit 6 bits
Tx7Bits	1	Transmit 7 bits
Tx8Bits	3	Transmit 8 bits

The other fields are one bit fields containing a 1 or a 0.

The field Regs.Write6 specifies the SIO chip's write register 6. It contains the 8 bit sync character.

The state of the speech hardware is reset to the default state whenever the file is closed.

The *ioctl(2)* system calls have the form:

```
#include "usr/include/sys/speech.h"
ioctl (filedes, code, arg)
union speechb *arg;
```

The applicable codes are:

TIOCGETP	Fetch the parameters associated with the device and store them in union referenced by arg. Fetching the parameters involves sensing the SIO chip to find the current state of read registers 0 and 1
TIOCSETP	Set the parameters associated with the device, fetching them from the union referenced by arg. The change occurs after any output in progress terminates
TIOCRSET	Reset the speech hardware to the default state immediately, aborting any output in progress.

The default state for the speech hardware is:

READ0	-	Undefined
READ1	-	Undefined
XmitRate	-	5
Write 4	-	0x00 (X1, Sync8Bit, SynModeEnable, NoParity)
Write 5	-	0xEA (Tx8Bits, DTR, TxEnable, RTS)
Write 6	-	0xAA (Sync pattern 10101010)

**SPEECH(4)****FILES**

/dev/speech.

**SEE ALSO**

*ioctl(2)*.



**TABLET(4)**

**NAME**

## tablet - graphic input device driver

## **DESCRIPTION**

`/dev/tablet` is a special file which allows graphic input into PNX. A common graphic input device is a tablet which consists of a movable puck resting on a pad. The position of the puck on the pad determines the (X, Y) co-ordinate pair the tablet sends to PNX. Most tablets have buttons. These tablets send the state of the buttons when they send co-ordinate information.

Only the *open(2)* and *close(2)* system calls are allowed with */dev/tablet*. See *screen(4)* or *wgread(2)* for information on how to receive information from the tablet. Window manager opens */dev/tablet* which starts sending puck X,Y co-ordinates at a rate determined by switch settings on the tablet. User programs should not use */dev/tablet* directly but would use system calls to an open window or */dev/screen* file to get the converted data.

The major and minor device number of `/dev/tablet` determine the type of graphic input device connected to PNX. PNX supports the following graphical input devices:

Standard tablet: major device number = 12  
minor device number = any /\* 0 = absolute  
1 = relative \*/

**Physical button 0 = left button**

**Physical button 1 = middle button**

**Physical button 2 = right button**

**Optional tablet:**

major device number = 6

minor device number bit significant as follows:

bits 0 - 5 = tablet address (usually 8)

bit 6 = relative tracking

bit 7 = device is a tablet

This minor device number must be >127 to signify a tablet. Minor device number 136 gives absolute tracking. Minor device number 200 gives relative tracking.

**Physical button 0 = yellow button**

**Physical button 1 = white button**

**Physical button 2 = blue button**

**Physical button 3 = green button**

**SEE ALSO**

*screen(4), wgrep(2), window(4), wprofile(5)*



TS(4)

**NAME**

TS - transport service interface

**DESCRIPTION**

The utility *mknet(1)* converts all the named files (agents and servers) to character special files and places all the special files in */lan*. The special files have a major device number of 11 and a minor device number in the range 0-255.

*mknet* assigns the minor device number when it creates the character special file corresponding to the agent or server and places an entry for the file in */lan/names* which is a file listing all the special files created by *mknet*, indexed by this minor device number.

The transport service is used by system calls to the special files created by *mknet*, that is, *open*, *close*, *read*, *write* and *ioctl*. An open call to a transport service special file passes the minor device number of the file to the transport service routines. The transport service entries use the minor device number to look up */lan/names* to get the unique address of the agent or server.

*mftp(1)* and *slaveft(1)* use the transport service routines to provide a PERQ to PERQ file transfer program (see Chapter 7 of *ICL PERQ Guide to PNX*).

*ioctl* calls to transport service devices have the general form:

```
#include <ecmauif.h>
ioctl (file, code, arg)
char *arg
```

The *filedes* is a file descriptor identifying the special file. *arg* is a pointer to an 8 bit value returned by the following codes:

ECMAGETTA	Returns the TSEL and network address of the remote process
ECMATASUM	Returns the number of transport service connections currently existing
ECMAIPSTATE	Returns a bit value in the first bit of the parameter area. Bit 1 value is: 1 if data available 0 if no data available
	The following code does not return any values:
ECMAXSTATE	Instructs the transport service that the next <i>write</i> call is an expedited write. Further <i>write</i> calls are standard unless again preceded by an <i>ioctl</i> ECMAXSTATE CALL

**FILES**

<i>/lan/mynname</i>	contains name of local machine
<i>/lan/names</i>	index to all transport service special files
<i>/lan/filename</i>	all transport service files created by <i>mknet</i> are placed in <i>/lan</i>

**SEE ALSO**

*mknet(1)*, *slaveft(1)*, *mftp(1)*, and section 2.3.4.2 of *ICL PERQ: Guide to PNX*.

**DIAGNOSTICS**

-1 returned if connection no longer exists

**TTY(4)****NAME**

`tty` - general terminal interface, under WMS

**DESCRIPTION**

This section describes both a particular special file, and the general nature of the terminal interface.

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

The file `/dev/tty` is made by issuing the command:

`/etc/mknod /dev/tty c 1 0`

The first terminal file open in a process becomes the *control terminal* for that process. In practice user's programs seldom open this file; it is opened by `init(8)` or by the Window Management System and becomes a user's input and output file. The control terminal plays a special role in handling quit or interrupt signals, as discussed below. The control terminal is inherited by a child process during a `fork(2)`, even if the control terminal is closed. The set of processes that thus share a control terminal is called a *process group*; all members of a process group receive certain signals together, see `^C` below and `kill(2)`.

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

Normally, terminal input is processed in units of lines. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not however necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information. There are special modes, discussed below, that permit the program to read each character as typed without waiting for a full line.

Certain ASCII control characters have special meaning. These characters are not passed to a reading program except in raw mode where they lose their special character. Also, it is possible to change these characters from the default (see below):

- `^Z`** May be used to generate an end of file from a terminal. When a `^Z` is received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the `^Z` is discarded. Thus if there are no characters waiting, which is to say the `^Z` occurred at the beginning of a line, zero characters will be passed back, and this is the standard end-of-file indication.
- `^C`** Is not passed to a program but generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location. See `signal(2)`
- `^\`** Generates the *quit* signal. Its treatment is identical to the interrupt signal except that unless a receiving process has made other arrangements it will not only be terminated but a core image file will be generated

**TTY(4)**

- ^P** Flips paging off/on for the current process, unless *sg\_length* in the *ttiocb* structure (see below) has been set to 0, when it is simply passed to the program unaltered
- ^S** Delays all output on the terminal until something is typed in (graphics output is also delayed)
- ^Q** Restarts output after **^S** without generating any input to a program
- ^R** Retypes the current line, from the last delimiter. Control characters are indicated as detailed in *inectl* below
- ^W** Erases the last word typed
- ^U** Erases the entire line up to the point of typing but not beyond a **^Z**

During input, erase and kill processing is normally done. By default, the character **DEL** (rubout) erases the last character typed, except that it will not erase beyond the beginning of a line or a delimiter (^Z (end of file), or *t\_brkc* (see below)). By default, the character **OOPS** (^U) kills the entire line up to the point where it was typed, but not beyond a ^Z. Both these characters operate on a keystroke basis independently of any backspacing or tabbing that may have been done. Either **OOPS** or **DEL** may be entered literally by preceding it by \; the erase or kill character remains, but the \ disappears. These two characters may be changed to others. If, however, they are changed to printing characters, the deletion does not appear on the screen, the erase character is printed as it is, and the kill character takes a new line. So, if you change them to # and @, then it looks exactly like the books say it should!

The **HELP** command key is treated specially in that pressing it generates a *help* signal to all processes with access to the window. The signal is normally ignored but may be trapped by a process. Note that the ASCII value to which the key would normally be mapped, for example **^G**, is not specially treated.

When a terminal window is deleted by the operator, a *hangup* signal is sent to all processes with the terminal as control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file on their input can terminate appropriately when hung up on.

Several *ioclt(2)* calls apply to the terminal aspects of windows. Most of them use the following structure, defined in *<sgtty.h>*:

```
struct sgttyb {
    char sg_ispeed;
    char sg_ospeed;
    char sg_erase;
    char sg_kill;
    int sg_flags;
    char sg_nldly;
    char sg_crdly;
    char sg_htdly;
    char sg_vtdly;
    char sg_width;
    char sg_length;
};
```

The *sg\_ispeed* and *sg\_ospeed* fields are not relevant to window terminals. The *sg\_erase* and *sg\_kill* fields of the argument structure specify the erase and kill characters respectively. (Defaults are **DEL** (rubout) and **OOPS** (^U).

**TTY(4)**

The *sg\_flags* field of the argument structure contains several bits that determine the system's treatment of the terminal:

PERQRAW	0010000	PERQ keystroke values
XTABS	0002000	Replace output tabs by spaces
INDCTL	0001000	Echo control characters as '^<char+0140>'
SCOPE	0000400	Enable neat erasing on VDUs
RAW	0000040	Raw mode: wake up on all characters, 8 bit interface
CRMOD	0000020	Map CR into LF; echo LF or CR as CRLF
ECHO	0000010	Echo
LCASE	0000004	Map upper case to lower on input
CBREAK	0000002	Return each character as soon as typed

PERQRAW allows the actual values generated by the PERQ keyboard to be read (for example, the CTRL key sets bit 7, see Appendix 6). If this mode is not set, keystrokes are translated to the usual ASCII values. Beware that in RAW mode, control keys are disabled and the ISO values are sent to the program. If PERQRAW is also on, the raw 8 bit values are sent, therefore, not all characters may be recognised when input, as all the above extra modes deal in 7 bit characters.

XTABS causes tabs to be replaced by the appropriate number of spaces on output. These spaces overwrite any text on the screen. Windows do their own tabbing and tabs are non-destructive, moving over text. XTABS switches this facility off. If the window is using a proportional font, a tab would normally move a certain number of character spaces (W character width) but if XTABS is on, a certain number of space characters are inserted, probably giving a different result.

INDCTL causes control characters to be echoed as "``<char+0140>". Normally, control characters are not echoed, to prevent terminals which are not really terminals from going into funny states. (If not set, non printing control characters are ignored by windows.)

In RAW Mode, every character is passed immediately to the program without waiting until a full line has been typed. With no other modes set, no erase or kill processing is done; the end-of-file indicator (^Z), the interrupt character (DEL) and the quit character (^X) are not treated specially. There is no echoing, and no replacement of one character for another; characters are a full 8 bits for both input and output. However, setting any of the additional modes below along with RAW will add these functions:

#### ECHO, LCASE, XTABS

Setting CBREAK along with RAW allows the stop and start characters (^S and ^Q by default) to function. Parity will affect these options.

CRMOD causes input carriage returns to be turned into new-lines.

ECHO causes automatic echoing of input characters. All echoed characters are 7 bits wide, even in RAW.

CBREAK is a sort of nearly raw mode. Programs can read each character as soon as typed, instead of waiting for a full line, but quit and interrupt work, and case-translation, XTABS and ECHO work normally. On the other hand there is no erase or kill, and no special treatment of ^Z.

Setting the *sg\_width* field to non-zero causes a newline to be output automatically when an input line exceeds this value.

The *sg\_length* field is used to give a screen length for paging. If the field is non-zero, output will be paged in pages of that number of lines. The next page is output when any character other than new-line (or RETURN), interrupt or quit is typed. Interrupt and quit work as expected, and newline (or RETURN) gives only the next line of output.

The delay fields are not relevant to window terminals.

The default flag settings for a window terminal are:

XTABS, SCOPE, CRMOD, ECHO

**TTY(4)**

Several *ioctl* calls have the form:

```
#include <sgtty.h>
ioctl(fd, code, arg)
struct sgttyb *arg;
```

The applicable codes are:

TIOCGETA	Fetch the parameters associated with the terminal, and store in the pointed-to structure
TIOCSETA	Set the parameters according to the pointed-to structure. The interface delays until output is quiescent, then throws away any unread characters, before changing the modes
TIOCGETP and TIOCSETP	These work as TIOCGETA and TIOCSETA respectively, but only on the first 6 bytes of the <i>sgttyb</i> structure (that is <i>sg_ispeed</i> to <i>sg_flags</i> ). Their retention is necessary for compatibility reasons
TIOCSETN	Set the parameters but do not delay or flush input. Switching out of RAW or CBREAK mode may cause some garbage input. This also only works on the first 6 bytes of the <i>sgttyb</i> structure

With the following codes the *arg* is ignored:

TIOCEXCL	Set the "exclusive-use" mode: no further opens are permitted until the file has been closed
TIOCNXCL	Turn off "exclusive-use" mode
TIOCHPCL	No effect

TIOCFLUSH All characters waiting in input or output queues are flushed

The following codes affect characters that are special to the terminal interface. The argument is a pointer to the following structure, defined in *<sgtty.h>*:

```
struct tchars {
    char t_intrc;      /* interrupt */
    char t_quitc;      /* quit */
    char t_startc;     /* start output */
    char t_stopc;      /* stop output */
    char t_eofc;       /* end-of-file */
    char t_brkc;       /* input delimiter (like nl) */
};
```

The default values for these characters are ^C, ^\, ^Q, ^S, ^Z, and -1. A character value of -1 eliminates the effect of that character. The *t\_brkc* character, by default -1, acts like a new-line in that it terminates a line, is echoed and is passed to the program. The stop and start characters may be the same, to produce a toggle effect. It is probably counterproductive to make other special characters (including erase and kill) identical.

The calls are:

TIOCGETC	Fetch the various special characters associated with the terminal, and store in the pointed-to structure
TIOCSETC	Set the special characters to those given in the structure

**TTY(4)**

The following codes refer to the function key settings. Whenever a function key is typed, PNX replaces the key value by the string assigned to the key. The effect is the same as a user actually typing the string. Each window has its own function key settings. The *ioctl* calls need *arg* to be a pointer to the following structure:

```
struct ttiopf {  
    char *pf_keys [4];  
};
```

The codes are:

**TIOCSTPF** Change the function key setting. *pf\_keys[0]* points to the string to assign to the key PF1 and so on

**TIOCGTPF** Fetch the current function key settings. *pf\_keys[0]* points to a buffer into which the system places the string assigned to PF1 and so on. The buffer for each key must be large enough to hold an 80 character string and the final '0'.

The total length of a PF string cannot exceed 80 characters

The following codes are only available to the superuser:

**TIOCDBON** Enable the microcode debugger

**TIOCDBOFF** Disable the microcode debugger

All *ioctl* call to */dev/console* (or any other file created by: */etc/mknod <name> C 0 0*) enables or disables the microcode debugger. If the debugger is disabled '^INS' has no special meaning. If enabled, '^INS' enters the debugger.

**FILES**

*/dev/tty*, any window special file.

**SEE ALSO**

*stty(1)*, *signal(2)*, *ioctl(2)*.



**WINDOW(4)****NAME**

window - general window interface, under WMS.

**DESCRIPTION**

This section describes the special files used for access to windows and the general nature of the window interface.

A window description is a file in the PNX filestore containing the parameters of a potential WMS window. It is not a special device but is a filestore file with a special format (see *wdesc(5)*) and is made using *mkwind(1)* or *mknod(2)*. A window is created by opening such a file or may be created directly by the Window Manager program.

Additionally, a special device interface exists to allow access to any created window (for example, to plot subsystems). The file */dev/windowN* shares access to the created window with internal number *N* (see *WIOCGETS* below). WMS supports up to 31 windows and *N* may be any number in the range 0..30.

The window special files are made by issuing the command:

```
/etc/mknod /dev/windowN c 9 N
```

Thus a process can gain access to a window by opening a window description file or by accessing a window special file. However the access is obtained, a process can use a window as a standard terminal (see *tty(4)*) or for graphics output (see *wrasop(2)*, *wline(2)* and *wdbyte(2)*) or tablet input (see *wgread(2)*). These facilities have modes which may be altered by *ioctl(2)* calls.

The following structure, defined in *sgwin.h* is used by *ioctl* for the requests defined here. Each request has a corresponding member of the union definition:

```
union sgwinb {
    int             sg_CuMode; /* for WIOCCMOD          */
    int             sg_CuFunc; /* for WIOCCFUN          */
    struct w_cpat  sg_CuPat;  /* for WIOCCPAT          */
    struct w_cpos  sg_CuPos;  /* for WIOCCPOS          */
    struct w_spcu  sg_SpCu;   /* for WIOCSPCU          */
    struct w_inm   sg_InMode; /* for WIOCENIN          */
    struct w_stat  sg_WStat;  /* for WIOCGETS          */
    struct w_tstat sg_WTStat; /* for WIOCGETT          */
    struct w_conf  sg_WConf;  /* for WIOCGCONF          */
};
```

The *ioctl* calls have the form

```
#include <sgwin.h>
ioctl (fd, request, argp)
union sgwinb *argp; /* ignoring sgtty etc */
```

In all cases a valid address of store equivalent to the union must be supplied even when no parameters are required.

The applicable requests are:

**WIOCCMOD** Set the cursor mode to be used when the window is selected, to the value in *sg\_Cumode*. Values may be:

**CMAUTO** Autotrack mode: the cursor tracks the tablet co-ordinates; the window is deselected if the tablet ceases to point to a visible area of the window

**CMXAUTO** Extended autotrack: the tablet may be placed in any position without deselection taking place

**CMON** On mode: the cursor does not track the tablet but appears at the position set by the user process; the tablet (and cursor) may be placed in any position

**WINDOW(4)**

- CMOFF** Off mode: as CMXAUTO but the cursor picture is switch off.
- A window is initialised to CMAUTO mode.
- WIOCCFUN** Set the cursor function to be used when the window is selected, to the value in *sg\_CuFunc*. This value also controls the background colour of the window.
- |                     |  |
|---------------------|--|
| <b>CFNORMAL</b>     | Window and cursor combined, logical OR           |
| <b>CFINVERT</b>     | As <i>CFNORMAL</i> , but inverted                |
| <b>CFCURSCOMP</b>   | Window and cursor combined, logical XOR          |
| <b>CFINVCURCOMP</b> | As <i>CFCURSCOMP</i> but inverted                |
| <b>CFBLACKHOLE</b>  | As <i>CFWHITEHOLE</i> but inverted               |
| <b>CFWHITEHOLE</b>  | Window and inverted cursor combined, logical AND |
- The functions *CFNORMAL*, *CFCURSCOMP* and *CFWHITEHOLE* give a window background colour white; the others give black. (Inverted if the basic screen colour is inverted.)
- WIOCCPAT** Set the cursor pattern to be used when the window is selected.
- ```
struct w_cpat {
    int     *CuPatp;      /* pattern data          */
    short   CuPX;        /* logical centre offset */
    short   CuPY;
};

typedef unsigned int cur_pat[128]; /* for size */
```
- The pattern area, pointed to by *CuPatp*, is 64 bits wide and 64 bits high, and must be double word aligned. The actual pattern is 57 bits wide, 64 bits high, aligned to the left (most significant) edge of the area. The logical centre of the pattern is specified by *CuPX* and *CuPY* as offsets from the top left of the pattern. If *CuPatp* is zero, the default cursor is set to be used.
- WIOCGPAT** Get the cursor pattern used by the window when selected. Structure *w\_cpat* is used with *CuPatp* pointing to a pattern area. The pattern is placed in the area and the offsets set; if the window is using the default cursor *CuPatp* is set to zero and no pattern is supplied.
- WIOCCPOS** Set the cursor position, relative to the window, to be used when the window is selected.
- ```
struct w_cpos {
    short   CuPosX;
    short   CuPosY; /* positions, window relative */
};

This command is only meaningful when the cursor mode is CMON
```

**WINDOW(4)**

**WIOSPCU** Enable special cursor to be displayed when the window is selected for graphics.

```
struct w_spcu {
    short CuSType      ; /* special cursor type*/
    short CuSSp1       ; /* spare*/
    short CuSP1X, CuSP1Y ; /* point 1*/
    short CuSP2X, CuSP2Y /* point 2*/
};
```

The field *CuSType* may take the values:

<b>CSTOFF</b>	Disable special cursor
<b>CSTRECT</b>	Enable, as rectangle based on the diagonal P1 to P2
<b>CSTLINE</b>	Line from P2 to P1
<b>CSTCROSS</b>	Cross centered on P1

P1 and P2 are window relative points, unless the following values are combined with logical OR into *CuSType*, in which case the corresponding co-ordinate is relative to the cursor:

<b>CSTX1R</b>	X of point 1 is cursor relative
<b>CSTY1R</b>	Y of point 1 is cursor relative
<b>CSTX2R</b>	X of point 2 is cursor relative
<b>CSTY2R</b>	Y of point 2 is cursor relative

The following complete values for *CuSType* are defined for standard use:

<b>CSTFRAME</b>	Wire frame, P1 and P2 relative to cursor
<b>CSTRBOX</b>	Rubber box, P2 fixed, P1 relative to cursor
<b>CSTRLINE</b>	Rubber line, P2 fixed, P1 relative to cursor
<b>CSTXHAIR</b>	Cross-hair on P1, relative to cursor

Note that standard cursor mode and special cursor mode may be independently enabled, except that the standard cursor mode determines the behaviour of the cursor which the special mode may be tracking. (As a special case, cursor mode CMOFF disables the standard cursor display, but any enabled special cursor tracks the cursor position as if the cursor were in CMXAUTO mode.)

The special cursor is clipped to the window boundary if the standard cursor mode is CMAUTO; otherwise it is clipped to the screen boundary

**WIOCENIN** Set the input mode to be used when the window is selected.

```
struct w_inm {
    char InMode      ; /* input mode */
    char TrFlags     ; /* which triggers */
    short TrTabs      ; /* tablet change */
    int TrTicks      ; /* clock ticks */
};
```

The field *InMode* may take the values:

<b>IMSAMPLE</b>	One input record is generated immediately on each call of <i>wgread(2)</i>
-----------------	--

<b>IMREQUEST</b>	One input record is queued for the window on the first (next) occurrence of a specified trigger condition after each subsequent call of <i>wgread(2)</i>
------------------	--

**WINDOW(4)**

IMEVENT	Input records are queued continuously for the window on every occurrence of a specified trigger condition after the mode is set. The number of events which may be queued is a configuration constant; if it is reached, further records are not queued
IMASEVENT	As IMEVENT but a SIGWINP signal is caused with each queued record (see <i>signal(2)</i> )
IMNOINT	If the window is selected by the operator, only the keyboard is attached to the window and not the graphic input device. The mode should be reset if graphic input is required (but if <i>wgread</i> is executed in this mode, the mode is changed to IMSAMPLE automatically)
	The initial default for a window is IMNOINT mode.
	The field <i>TrFlags</i> specifies which trigger events are of interest.
TRPUCK	Any puck button/pen switch change of state (since mode set or subsequent most recent input record generated)
TRTAB	Any change of tablet co-ordinates, in any dimension by at least the number of pixels specified in the <i>TrTabs</i> field (since mode set or subsequent most recent input record generated)
TRTICK	Elapsed time (counted only while the window is selected) specified by the <i>TrTicks</i> field in units of the internal clock (60 Hz) (since mode set or subsequent most recent input record)
TRKEY	This trigger occurs if data is available to a standard <i>read(2)</i> call. Once the trigger has occurred it is inhibited until a <i>read(2)</i> is executed or the mode reset
TRSEL	This trigger occurs when the selection state of the window changes to <i>selected</i> or <i>deselected</i>
	The trigger conditions, except TRSEL, are only evaluated while the window is selected for graphical input.
WIOCGINP	Get the input mode currently set for the window; all the fields of structure <i>w_inm</i> are set.
WIOCGETS	Get the current status of the window.  struct w_stat { short WSX ; /* X position on screen */ short WSY ; /* Y .. */ short WSWidth; /* width in pixels */ short WSHeight ; /* height .. */ short WSRank ; /* relative rank */ char WSCuMode ; /* cursor mode set */ char WSInMode ; /* input mode set */ short WSFlags ; /* user and some system flags */ char WSCuFunc ; /* cursor function in use */ char WSID ; /* internal window number */ };

**WINDOW(4)**

```

/* values for WSFlags field */
```

WSBORDER	1	/* has border */
WSTITLE	2	/* has title line */
WSUNIQUE	4	/* is shareable, unique instance */
WSDONC	8	/* to be deleted on close */
WSIMAGE	0x0100	/* is stored in image format */
WSSELECT	0x0200	/* is selected for keyboard */
WSNOUPD	0x0400	/* screen update temporarily disabled by user */
WSOBSURED	0x1000	/* not fully displayed on screen */
WSDELETED	0x8000	/* has been deleted by operator */

**WIOCGGETT** Get additional status concerning text display

```

struct w_tstat {
```

char WTWidht	/* window width in characters */
char WTHeight	/* window height in character */
char WTTtextX	/* text cursor, character number in line */
char WTTtextY	/* text cursor, text line number */
char WTFontD	/* internal font number, 0=standard font */
short WTFontHt	/* window font character pixel height */
short WTFontBase	/* window font character pixel base */
short WTFontWid	/* window font character pixel width. width is that of 'W' character */
char WTMargin	/* character left margin pixel position */
char WTLead	/* extra pixel lines between text lines. These are in- serted above the text line */
char WTSp1,WTSp2	/* spare */

};

**WIOCNOUPD** Stops WMS from updating the on-screen image of the window after any graphics operation (*wdbyte(2)*, *wline(2)*, or *wrasop(2)*). Enables build up of an entire image; may have some effect on performance. Also stops the WMS from updating screen image after terminal *read(2)* or *write(2)* operations except:

## WINDOW(4)

- 1 Whenever output is stopped by the operator or by terminal paging an immediate refresh is carried out
- 2 *read(2)* call causes an immediate refresh and cancellation of the mode unless echoing is disabled

**WIOCDOUP** Refresh the on-screen image of a window from the off-screen image and restart automatic refreshment after any graphics or text operation.

**WIOCDONC** Set window to be removed from screen automatically when closed by all users.

**WIOCPONC** Set window to be preserved on screen when closed by all users unless deleted by the operator

**WIOCGCONF** Get window system configuration information. The information is not specific to any window.

```
struct w_conf {  
    char    WCFlags      ;  
    char    WCWMEsc     ; /* WM Escape raw key value */  
    char    WCTabMap [4] ; /* map tablet physical  
                           buttons to logical */  
};
```

These bits are defined by name in **WCFlags**:

**WCWMSRUN**, defined by name **WCFlags**, is set if at least one window is active. This list usually denotes that WMS is initialised and running.

Entry *i* of **WCTabMap** defines the value of logical button(s) (see *wgread(2)*) which are set when physical button *i* is depressed on the tablet device (see *tablet(4)*).

## SEE ALSO

*ioctl(2), tty(4), tablet(4), wdbyte(2), wline(2), wrasop(2), wgread(2)*.

**VP(4)****NAME**

vp - ICL 6203 Electrostatic Printer/Plotter

**DESCRIPTION**

The file `/dev/vp` gives printing access to the ICL 6203 Electrostatic Printer/Plotter when it is connected on the GPIB. The printer should be connected to the GPIB on GPIB address 1 and should be declared to PNX as follows:

`/etc/mknod /dev/vp c 6 35`

The printer should be set up as follows:

PRINT mode

132 character/line

64 lines/page

**FILES**

`/dev/vp`

**SEE ALSO**

*gpib(4)*



**Z80(4)****NAME**

Z80 - special file for loading Z80 firmware (only applies to PERQ2)

**DESCRIPTION**

The file `/dev/Z80` is opened by the system program *tekload* to load the Z80 firmware on the EIO board.

The interface is via the system calls *open(2)*, *lseek(2)* and *write(2)*.

*open(2)* returns a file identifier to be used in subsequent system calls

*lseek(2)* specifies two things:

- 1 If bit  $2^{16}$  is not set, then specifies the address in Z80 RAM the next buffer is to be written to
- 2 If bit  $2^{16}$  is set, specifies that the Z80 processor is to be started with a *write registers* command at the address the previous *lseek* specified (modulo  $2^{16}$ )



**A.OUT(5)****NAME**

a.out - assembler and link editor output

**SYNOPSIS**

```
#include <a.out.h>
```

**DESCRIPTION**

*a.out* is the default output file of the assembler *as*(1) and link editor *ld*(1). *as*(1) passes its output to *ld*(1). *ld*(1) makes *a.out* executable if there were no errors or unresolved external references. *ld*(1) produces a file with four sections in the following order: a header, the program and data code, relocation information and a symbol table. The last two may be empty if the program is loaded with the -s option or if the symbols have been removed by *strip*(1).

*ld*(1) uses the information in *a.out.h* to determine the layout of the header, the symbol table entries and the flag values to distinguish symbol types. *a.out.h* includes the following information:

```
#define SYMSIZE 32
struct exec {
    int     a_magic;           /* a.out header */
    unsigned a_text;           /* magic number */
    unsigned a_data;           /* size of text segment */
    unsigned a_bss;            /* size of initialised data */
    unsigned a_syms;           /* size of uninitialised data */
    unsigned a_entry;          /* size of symbol table */
    unsigned a_stamp;          /* entry point */
    unsigned a_flag;           /* always set to -1. Other version
                                number stamps not supported */
};                                /* relocation info stripped */

#define A_MAGIC3 0411             /* separated I&D */
#define FORMAT "%.8x"             /* format to print a value */
#define STABTYPES 0340            /* mask to check that types
                                actually exist */

#define BADMAGIC(X) (x.a.magic!=A_MAGIC3) /* ld(1) calls the macro
                                         BADMAGIC(X) to check
                                         that the file is an object file.
                                         X must not be a function */

struct nlist {
    char    n_name[SYMSIZE];    /* symbol table entry */
    char    n_type;              /* symbol name */
    char    n_sect;              /* type flag */
    short   n_desc;              /* section id */
    /* unused */                /* unused */
    short   n_desc;              /* used by compilers to place
                                information about name list
                                entries for use by sdb*/
    unsigned n_value;            /* entries for use by sdb*/
};                                /* value */

/* values for type flag */
#define N_UNDF 0                 /* undefined */
#define N_ABS 02                  /* absolute */
#define N_TEXT 04                  /* text symbol */
#define N_DATA 06                  /* data symbol */
#define N_BSS 010                 /* bss symbol */
#define N_FUNC 012                 /* text (label) symbol */
#define N_ERC 014                  /* local data symbol used by ERCC
                                Fortran */
```

**A.OUT(5)**

```

#define N_REG      030          /* register symbol */
#define N_VNO      032          /* version number symbol */
#define N_TYPE     036          /* file name symbol */
#define N_FN       036          /* external bit */
#define N_EXT      01           /* global sym: name,,type,0 */
#define N_GSYM    0040          /* procedure name (f77 kludge): name
                                ,,,0 */
#define N_FNAME   0042          /* procedure: name,,linenumber,
                                address */
#define N_STSYM   0046          /* static symbol: name,,type,
                                address */
#define N_LCSYM   0048          /* .lcomm symbol: name,,type,
                                address */
#define N_RSTR    0060          /* begin structure: name,,, */
#define N_RSYM    0100          /* register sym: name,,register,
                                offset */
#define N_RSAVED  0102          /* register save area: ,,which,
                                offset */
#define N_SLINE   0104          /* source line: ,,linenumber,
                                address */
#define N_ESTR    0120          /* end structure: name,,, */
#define N_SSYM    0140          /* structure elt: name;,type,,
                                structoffset */
#define N_SO      0144          /* source file name: name,,,
                                address */
#define N_BENUM   0160          /* begin enum: name,,, */
#define N_LSYM    0200          /* local sym: name,,type,offset */
#define N_SOL     0204          /* #line source filename: name,,,
                                address */
#define N_ENUM    0220          /* enum element: name,,,value */
#define N_PSYM    0240          /* parameter: name,,type,offset */
#define N_ENTRY   0244          /* alternate entry: name,,,
                                linenumber,address */
#define N_EENUM   0260          /* end enum: name,,, */
#define N_LBRAC   0300          /* left bracket: ,,nesting
                                level,address */
#define N_RBRAC   0340          /* right bracket: ,,nesting
                                level,address */
#define N_BCOMM   0342          /* begin common: name,,, */
#define N_ECOMM   0344          /* end common: name,,, */
#define N_ECOML   0348          /* end common (local name): ...,
                                address */
#define N_STRU    0374          /* 2nd entry for structure: str tag
                                ,,, length */
#define N LENG   0376          /* additional entry with length:
                                ... */

```

In the header, the sizes are given in bytes and for *ld(1)* output are always a multiple of 512.

The size of the header is not included in any of the other sizes.

For efficiency when executing a file, the text and data sections each begin on 512 byte boundaries in the file.

The output of the link editor includes additional data in the header, as follows:-

```

struct {
    int    cktext;
    int    ckdata;
    int    time;           /* timestamp */
    char   name [16];
    int    version;
    int    page_size;      /*maximum size of text pages */
}

```

**A.OUT(5)**

This contains data which is displayed by the *file(1)* program.

The timestamp indicates when loader made the file.

When an *a.out* file is loaded into store for execution, areas of store are allocated to hold the text, the data, and the stack for the process. If the *a.out* file is already in use, a second copy of the text is not brought into store; instead, the existing copy is shared among all users.

In a process, the stack starts below address zero and is extended downwards automatically; stack addresses are always negative. The data starts at address zero and always has positive addresses; the size of the data region may be changed by *brk(2)*.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader *ld* as the name of a common region whose size is indicated by the value of the symbol.

**SEE ALSO**

*as(1), file(1), ld(1), nm(1)*



**ERRFILE(5)****NAME**

errfile - error-log file format

**DESCRIPTION**

When hardware errors are detected by the system, an error record is generated and passed to the error-logging daemon for recording in the error log for later analysis.

The format of an error record depends on the type of error that was encountered. Every record, however, has a header with the following format:

```
struct errhdr {
    int      e_type;      /* record type */
    int      e_len;       /* bytes in record time (inc hdr) */
    time_t   e_time;     /* time of day */
};
```

The permissible record types are as follows:

```
#define E_GOTS  010  /* Start for UNIX 3.0 */
#define E_GORT  011  /* Start for UNIX/RT */
#define E_STOP  012  /* Stop */
#define E_TCHG  013  /* Time change */
#define E_CCHG  014  /* Configuration change */
#define E_BLK  020  /* Block device error */
#define E_STRAY 030  /* Stray interrupt */
#define E_PRTY  031  /* Memory parity */
#define E_STR  032  /* Error string */
```

E\_GORT, E\_STOP, E\_TCHG, E\_CCHG, E\_STRAY and E\_PRTY are not used by PNX.

Some records in the error file are of an administrative nature. These include the start up record that is entered into the file when logging is activated, the stop record that is written if the daemon is terminated "gracefully", and the time-change record that is used to account for changes in the system's time-of-day. These records have the following formats:

```
struct estart {
    struct errhdr  e_hdr;      /* record header */
    int            e_cpu;      /* CPU type */
    int            e_mmr3;     /* contents mem mgmt reg 3 */
    long           e_syssize;  /* 11/70 system memory size */
    int            e_bconf;    /* block dev configuration */
};

struct eend {
    struct errhdr  e_hdr;      /* record header */
};

struct etimchg {
    struct errhdr  e_hrd;     /* record header */
    time_t         e_ntime;    /* new time */
};
```

Stray interrupts cause a record with the following format to be logged in the file:

```
struct estray {
    struct errhdr  e_hdr;      /* record header */
    physadr        e_saddr;    /* stray loc or device addr */
    int            e_sbacty;   /* active block devices */
};
```

**ERRFILE(5)**

Memory subsystem error on 11/70 processors cause the following record to be generated:

```
struct eparity {
    struct errhdr e_hdr;          /* record header */
    int           e_parreg[4];     /* memory subsys registers */
};
```

Error records for block devices have the following format:

```
struct eblock {
    struct errhdr e_hdr;          /* record header */
    dev_t         e_dev;           /* "true" major + minor dev no */
    physadr      e_reglloc;       /* controller address */
    int           e_bacty;         /* other block I/O activity */

    struct iostat {
        long        io_ops;          /* number read/writes */
        long        io_misc;         /* number "other" operations */
        unsigned    io_unlog;        /* number unlogged errors */
    };
    int           e_bflags;         /* read/write, error, etc */
    int           e_cyllopp;        /* logical dev start cyl */
    daddr_t      e_bnum;           /* logical block number */
    unsigned    e_bytes;           /* number bytes to transfer */
    long         e_memadd;         /* buffer memory address */
    unsigned    e_rtry;            /* number retries */
    int           e_nreg;           /* number device registers */
};
```

*eblock* is filled as follows

e_dev	bp → b_dev
e_reglloc	0
e_bacty	0
e_stats.io_ops	0
e_stats.io_misc	0
e_stats.io_unlog	0
e_bflags	bp → _flags   B_ERROR
e_cyllopp	0
e_bnum	bp → b_blkno
e_bytes	bp → b_bcount
e_memadd	bp → b_un.b_addr
e_rtry	bp → b_errcnt
e_nreg	0

E\_STR uses the following structure:

```
struct estring {
    struct errhdr e_hdr;          /* record header */
    char e_ch[1];                 /* error string */
};
```

*e\_ch* points to a string of arbitrary length, *e\_ch[0]* being the first character, *e\_ch[1]* the second and so on. The string terminates with '*'0*'. To use the structure, *read* into a large character array, assign its address to a pointer to struct *e\_string* and use the pointer to access the string.

The following values are used in the *e\_bflags* word:

#define E_WRITE 0	/* write operation */
#define E_READ 1	/* read operation */
#define E_NOIO 02	/* no I/O pending */
#define E_PHYS 04	/* physical I/O */
#define E_MAP 010	/* Unibus map in use */
#define E_ERROR 020	/* I/O failed */

**ERRFILE(5)**

The "true" major device numbers that identify the failing device are as follows:

#define	RK0	0
#define	RP0	1
#define	RF0	2
#define	TM0	3
#define	TC0	4
#define	HP0	5
#define	HT0	6
#define	HS0	7

**SEE ALSO**

*printerr(1)*



**FILSYS(5)****NAME**

filsys, fblk, ino - format of file system volume

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/filsys.h>
#include <sys/fblk.h>
#include <sys/inc.h>
```

**DESCRIPTION**

Both the fixed disc and the floppy disc, when used as file system volumes, have a common format for certain vital information. Every such disc is divided into a number of 512 byte blocks. Some parts of the discs are reserved; for example, cylinder zero of the fixed disc contains the PNX bootstrap.

Block 1 is the superblock. The layout of the superblock is defined by the include file <sys/filsys.h>. Two versions of the superblock are defined; the first describes the block on disc, the second describes the in-store format used by the kernel. These are different due to the differing architectures of the PERQ and the PDP11 machines.

```
struct PDPSUPER {
    /* This is the PDP11
     * superblock */
    /* size in blocks of i-list */
    /* size in blocks of entire
     * volume */
    /* number of addresses in
     * s_free */
    /* free block list */
    /* number of i-nodes in
     * s-inode */
    /* free i-node list */
    /* lock during free list
     * manipulation */
    /* lock during i-list
     * manipulation */
    /* super block modified flag */
    /* mounted read-only flag */
    /* last superblock update */

    /* remainder not maintained by this version of the system */
    /* total free blocks */
    /* total free i-nodes */
    /* interleave factor */
    /* interleave factor */
    /* file system name */
    /* file system pack name */

};

struct filsys {
    /* the PERQ working superblock */
    /* size in blocks of i-list */
    /* size in blocks of entire
     * volume */
    /* number of addresses in s-free */
    /* free block list */
    /* number of i-nodes in s-inode */
    /* free i-node list */
    /* lock during free list
     * manipulation */
    /* lock during i-list
     * manipulation */
    /* superblock modified flag */
    /* mounted read-only flag */
    /* last superblock update */

    /* isize */
    /* fsize */
    /* nfree */
    /* free[NICFREE] */
    /* ninode */
    /* inode[NICINOD] */
    /* flock */
    /* ilock */
    /* fmod */
    /* ronly */
    /* time */
};
```

**FILSYS(5)**

```

/* remainder not maintained by this version of the system */

daddr_t      s_tfree;          /* total free block */
ino_t        s_tinode;         /* total free inodes */
short       s_m;              /* interleave factor */
short       s_n;              /* interleave factor */
char        s_fname[6];        /* file system name */
char        s_fpack[6];        /* file system pack name */

};


```

*s\_isize* is the address of the first block after the i-list, which starts just after the superblock, in block 2. Thus the i-list is *s\_isize* - 2 blocks long. *s\_size* is the address of the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block addresses; if an impossible block address is allocated from the free list, or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The *s\_free* array contains, in *s\_free[1]*, ..., *s\_free[s\_nfrees - 1]*, up to NICFREE free block numbers. NICFREE is a configuration constant. *s\_free[0]* is the block address of the head of a chain of blocks constituting the free list. The layout of each block of the free chain as defined in the include file <sys/fblk.h> is:

```

struct fblk {
    int           df_nfrees;
    daddr_t      df_free[NICFREE];
};


```

The fields *df\_nfrees* and *df\_free* in a free block are used exactly like *s\_nfrees* and *s\_free* in the superblock. To allocate a block: decrement *s\_nfrees*, and the new block number is *s\_free[s\_nfrees]*. If the new block address is 0, there are no blocks left, so give an error. If *s\_nfrees* became 0, read the new block into *s\_nfrees* and *s\_free*.

To free a block, check if *s\_nfrees* is NICFREE; if so, copy *s\_nfrees* and the *s\_free* array into it, write it out, and set *s\_nfrees* to 0. In any event, set *s\_free[s\_nfrees]* to the freed block's address and increment *s\_nfrees*. *s\_inode* is the number of free i-numbers in the *s\_inode* array.

To allocate an i-node: if *s\_inode* is greater than 0, decrement it and return *s\_inode[s\_ninode]*. If it was 0, read the i-list and place the numbers of all free inodes (up to NICINOD) into the *s\_inode* array, then try again.

To free an i-node, provided *s\_ninode* is less than NICINOD, place its number into *s\_inode[s\_ninode]* and increment *s\_ninode*. If *s\_ninode* is already NICINOD, do not bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the i-node is really free or not is maintained in the i-node itself.

*s\_flock* and *s\_ilock* are flags maintained in the core copy of the file system while it is mounted and their values on disc are immaterial. The value of *s\_fmod* on disc is likewise immaterial; it is used as a flag to indicate that the superblock has changed and should be copied to the disk during the next periodic update of file system information. *s\_ronly* is a write-protect indicator; its disk value is also immaterial.

*s\_time* is the last time the superblock of the file system was changed. During a reboot, *s\_time* of the superblock for the root file system is used to set the system's idea of the time.

The fields *s\_tfree*, *s\_tinode*, *s\_fname* and *s\_fpack* are not currently maintained.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. I-nodes are 64 bytes long, so 8 of them fit into a block. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. The format of an i-node as given in the include file <sys/ino.h> is:

**FILSYS(5)**

```

struct dinode {
    unsigned short di_mode;           /*inode structure as on discblock*/
    short          di_nlink;          /* mode and type of file*/
    short          di_uid;            /* number of links to file */
    short          di_gid;            /* owner's user id */
    off_t          di_size;           /* owner's group id */
    char           di_addr[40];        /* number of bytes in the file */
    time_t         di_atime;          /* disc block addresses */
    time_t         di_mtime;          /* time last accessed */
    time_t         di_ctime;          /* time last modified */
};

#define INOPB 8                      /* time created */

/* the 40 address bytes: 39 used; 13 addresses of 3 bytes each */

```

*di\_mode* tells the kind of file; it is encoded identically to the *st\_mode* field of *stat(2)*. *di\_nlink* is the number of directory entries (links) that refer to this i-node. *di\_uid* and *di\_gid* are the owner's user and group IDs. *size* is the number of bytes in the file. *di\_atime* and *di\_mtime* are the times of last access and modification of the file contents (read, write, or create) (see *times(2)*); *di\_ctime* records the time of last modification to the i-node or to the file, and is used to determine whether it should be dumped.

Special files are recognised by their modes and not by i-number. A block-type special file is one which can potentially be mounted as a file system; a character-type special file cannot, though it is not necessarily character-oriented. For special files, the *di\_addr* field is occupied by the device code (see *types(5)*). The device codes of block and character special files overlap.

Disc addresses of plain files and directories are kept in array *di\_addr* packed into three bytes each. The first ten addresses specify service blocks directory. The last three addresses are singly, double, and triply indirect and point to blocks of 128 block pointers. Pointers in indirect blocks have the type *daddr\_t* (see *types(5)*).

For block *b* in a file to exist, it is not necessary that all blocks less than *b* exist. A zero block number either in the address words of the i-node or in an indirect block indicates that the corresponding block has never been allocated. Such a missing block reads as if it contained all zero words.

**SEE ALSO**

*dcheck(1), fsck(1), icheck(1), mount(1), stat(2), dir(5), types(5)*.



**MENUFILE(5)****NAME**

*menufile* - input file for *buildtext(3)*

**DESCRIPTION**

*menufile* must have the following format:

Title	/* an optional string of characters excluding ^	*/
Command number	/* an unsigned integer	*/
Command	/* string of characters excluding control	*/

The title is optional but if present should end with a new line delimiter.

A command line consists of a command number and command text. Each command line must start with a number. Command numbers and command text may be separated by spaces or tabs. Each complete command line must end with a new line delimiter.

**SEE ALSO**

*buildtext(3)*.



**PLOT(5)****NAME**

plot.h - header file for plot functions

**SYNOPSIS**

```
#include plot.h
```

**DESCRIPTION**

The include file *plot.h* is used by the *plot* and *splot* programs (see *plot(1)*) to define the format of any file stored by *plot*. The file contains the following type definitions:

```
#define NULL      0
#define V80max    132

struct plottable {
    int      plotX;           /* X, Y dimensions of, and pointer to, */
    int      plotY;           /* window image */
    char    *plotrec;
};

union pblock {
    struct plhdr {
        char    plid[5];        /* PLOT null X, Y flags; */
        int     plX;             /* header for plot file */
        int     plY;
        int     plflg;
    };
    char   pBuffer[512];
};
```



**SCCSFILE(5)****NAME**

sccsfile - format of SCCS file

**DESCRIPTION**

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

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 011). This character is hereafter referred to as *the control character* and will be represented graphically as @. Any line described below which is not depicted as beginning with the control character is prevented from beginning with the control character.

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

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

*Checksum*

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

@hDDDDD

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

*Delta table*

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

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

.

.

@c <comments>...

.

.

@e

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

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

## SCCSFILE(5)

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

The @e line ends the delta table entry.

### User names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines @u and @U. An empty list allows anyone to make a delta.

### Flags

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

@f <flag> <optional text>

The following flags are defined:

```
@f t  <type of program>
@f v  <program name>
@f i
@f b
@f m <module name>
@f f  <floor>
@f c  <ceiling>
@f d  <default-sid>
@f n
@f j
@f l  <lock-releases>
@f q  <user defined>
```

The t flag defines the replacement for the %Y% identification keyword. The v flag controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program. The i flag controls the warning/error aspect of the "No id keywords" message. When the i flag is not present, this message is only a warning; when the i flag is present, this message will cause a "fatal" error (the file will not be gotten, or the delta will not be made). When the b flag is present the -b keyletter may be used to the *get* command to cause a branch in the delta tree. The m flag defines the first choice of the replacement text of the %M% identification keyword. The f flag defined the "floor" release; the release below which no deltas may be added. The c flag defines the "ceiling" release; the release above which no deltas may be added. The d flag defines the default SID to be used when none is specified on a *get* command. The n flag causes *delta* to insert a "null" delta (a delta that applies no changes) in those releases that are skipped when a delta is made in a new release (for example, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the n flag causes skipped releases to be completely empty. The j flag causes *get* to allow concurrent edits of the same base SID. The l flag defines a list of releases that are locked against editing (*get(1)* with the -e keyletter). The q flag defined the replacement for the %Q% identification keyword.

### Comments

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

**SCCSFILE(5)***Body*

The body consists of text lines and control lines. Text lines don't begin with the control character, control lines do. There are three kinds of control lines: *insert*, *delete*, and *end*, represented by:

**@I DDDDD**  
**@D DDDDD**  
**@E DDDDD**

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

**SEE ALSO**

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

Bonanni, L.E. and Salemi, C.A., *Source Code Control System User's Guide*



**SPOOLGOV(5)****NAME**

/etc/spoolgov

**DESCRIPTION**

*/etc/spoolgov* contains a series of text entries, four per spoolqueue, each entry separated by white space characters. The entries for each spoolqueue are as follows:

spoolqueue    despooler    ACTIVE    CLOSED

where *spoolqueue* is the name of the spoolqueue (maximum 39 characters) for example, */usr/spool/lpd*. *dspooler* is the name of the despatcher (maximum 39 characters) for example, */usr/include/lpdel*. ACTIVE indicates whether the despooler is currently passing a file to a printer (1 = active, 0 = not active). CLOSED indicates whether the printer is available (1 = closed, 0 = not CLOSED).

Comments can be inserted between /\* and \*/ at the front and back of the file and in between groups of four entries.



**WCURS(5)****NAME**

wcurs - WMS cursor patterns

**DESCRIPTION**

The directory */etc/wcurs* contains a set of files which contain cursor patterns to be used by the WMS. It is accessed when the WMS is initialised. The following files are searched for:

<i>/etc/wcurs/sel</i>	- default selected state cursor
<i>/etc/wcurs/uncom</i>	- uncommitted state cursor
<i>/etc/wcurs/move</i>	- moving cursor
<i>/etc/wcurs/kill</i>	- deleting cursor
<i>/etc/wcurs/menu</i>	- pop up menu cursor
<i>/etc/wcurs/KBRQ</i>	- keyboard cursor
<i>/etc/wcurs/plot</i>	- plot cursor
<i>/etc/wcurs/pop</i>	- pop/push cursor
<i>/etc/wcurs/create</i>	- create (mark top left corner) cursor
<i>/etc/wcurs/dc</i>	- create (mark bottom right corner) cursor
<i>/etc/wcurs/size</i>	- change size cursor

If any file is not present, a default is substituted by the WM program.

Each file must have the following format (defined in *sgwin.h*):

```
struct wf_cpat { /* see WIOCCPAT in window (4) */
    cur_pat fCuPat;
    short   fCuPX;
    short   fCuPY;
};
```

A default set is supplied with the WMS. Any or all may be changed by the user.

**SEE ALSO**

*winit(1)*, *window(4)*.



**WDESC(5)****NAME**

wdesc - format of window descriptor file

**SYNOPSIS**

```
#include <wdesc.h>
```

**DESCRIPTION**

To create a window descriptor, that is, a special file containing a description of the properties of a required window virtual device, it is first necessary to create a file specifying the properties. The file should contain one record in the format, obtained from the include file *wdesc.h*:

```
struct wdesc {                                /* structure for window      */
    int          win_dclass      ;             /* descriptor file           */
                                                /* descriptor type,          */
                                                /* zero for this             */
    struct ClipCtl   win_screenarea ;         /* defined in wuser.h     */
    short        win_Rank        ;             /* relative to front,        */
                                                /* +ve away from user       */
    char         win_CuMode      ;             /* values from sgwin.h   */
    char         win_CuFunc      ;             /* values from sgwin.h   */
    short        win_Flags        ;             /* value defined below      */
    char        *  win_Title       ;             /* pointer to string         */
    char        *  win_Font        ;             /* pointer to string         */
    char        *  win_CuPat      ;             /* pointer to string         */
    int          win_Id          ;             /* window manager            */
                                                /* interface only             */
};

/* In the file, the structure is at the start, followed by strings, if any. */
/* String pointers are character offsets within the file. Each string must be terminated by
 \0. */

/* values for win_Flags */
WDBORDER 1
WDTITLE 2
WDUNIQUE 4
WDDONC 8 /* delete on close */

/* mode for mknod system call */

Use value IFWIN from sys/inode.h as follows:
mknod (file, IFWIN,0)
file is converted to a window descriptor special file.
```

**SEE ALSO**

*mkwind(1)*, *mknod(2)*



**WPROFILE(5)****NAME**

wprofile - WMS initialisation profile

**DESCRIPTION**

WMS initialisation profile consists of a set of files which, together with *wcurs(5)*, define the visible aspect of the WMS.

*Help files*

*/etc/wdev/sevhelp* and */etc/wdev/comhelp* hold text information displayed in the help windows described by the window descriptor special files */etc/wdev/genwnd* and */etc/wdev/comwnd* respectively. *sevhelp* and *comhelp* are displayed by the keyboard commands, **h** or **H**, and menu **Help** command respectively.

*Genwnd* and *comwnd* are generated by *mkwind(1)* from the text files */etc/wetc/gendsc* and */etc/wetc/comdsc* respectively.

*Menu files*

*/etc/wdev/commenu* contains the window manager menu commands, format as defined by *menufile(5)*.

*Error Message file*

*/etc/wdev/wmerror* contains the WMS error messages, held in text format, one message per line.

*Configuration file*

The file */etc/wprofile* or *wprofile* (in the users current directory) contains a specification of WMS options to be applied at WMS initialisation. (*Wprofile* is searched for first, and if not found the directory */etc* is used).

Options include:

Basic screen colour

WM Escape key value

Soft puck buttons

Initial descriptor open commands

*Wprofile* must be created as a text file using the key word and parameter statement formats:

OPEN      *filename*  
SCREEN    INVERT|WHITE  
WMESC    octal raw key value

A      <n>

B      <n>

C      <n>

D      <n>

where A, B, C and D represents the logical puck buttons, and, <n> is a physical button number, where,  $0 \leq n \leq 3$ . The statement A2, for example, states that physical button 2 is to be mapped to logical button A.

**WPROFILE(5)**

The default settings are:

```
SCREEN  WHITE
WMESC  215    /* that is raw ^RETURN */
A      0
B      1
C      2
C      3
```

**SEE ALSO**

*winit(1), wcurs(5).*

**GETTY(8)****NAME**

*getty* - set typewriter mode

**SYNOPSIS**

/etc/getty [char]

**DESCRIPTION**

*getty* is invoked by *init(8)* for each teletype terminal connected to the PERQ. *getty* reads the user's login name entered in response to the login prompt, and calls *login(1)* with the name as argument.

*init* calls *getty* with a single character argument taken from the *ttys(5)* file entry for */dev/console*. This argument determines the 'login:' greeting message.

The following arguments from *ttys(5)* file are understood:

- 0 Cycles through 300-1200-150-110 baud. Useful as a default for dial up lines accessed by a variety of terminals
- Intended for an online Teletype Model 33, for example an operator's console
- 1 Optimised for a 150 baud Teletype model 37
- 2 Intended for an online 9600 baud terminal for example in Textrimix 4104
- 3 Starts at 1200 baud, cycles to 300 and bulk. Useful with 212 datasets where most terminals run at 1200 speed
- 4 Useful for online console DECwriter (LA36)
- 5 Same as B but starts at 300
- 6 Intended for on-line 9600 baud terminal. Identical to argument type 2 except that even parity is used and the line and the line length and width are set to 24 and 80 respectively (both zero for argument type 2). This enables the screen paging facility using ^P. In addition, scope mode deletion is set (that is, deletions are performed on the text on the screen). This type is better suited to a glass teletype.

**SEE ALSO**

*login(1)*, *ioctl(2)*, *ttys(5)*, *init(8)*.



**INIT(8)****NAME**

*init*, *rc* - system initialisation

**SYNOPSIS**

*/etc/init*  
*/etc/rc*

**DESCRIPTION**

*init* is invoked as the last step of the PNX boot procedure. Its role is to supervise the processes running under PNX.

When *init* is first executed, the PERQ display and keyboard (*/dev/console*) is opened for reading and writing. *init* runs *fsck(1)* and if no errors are detected, *init* enters multi user mode.

The first action taken is to invoke a shell to obey the file */etc/rc*. This command file performs housekeeping functions like removing temporary files and starting daemons.

*init* then forks and so creates the login process for the PERQ user. File descriptors 0, 1, and 2 are set up for the standard input, output, and error routes. *getty(8)* initialises the PERQ according to the parameter in */etc/ttys*, and then invokes *login(1)* to log in the user and execute the shell.

Ultimately, the shell will terminate because of an end-of-file caused by the user typing ^Z. The main path of *init*, which has been waiting for such an event, wakes up and removes the entry from the file *utmp*, which records the user of the PERQ, and makes an entry in the file */usr/adm/wtmp*, which maintains a history of logins and logouts. *getty* is then invoked to restart the login process.

*init* catches the hangup signal SIGHUP and interprets it to mean that the system should be brought from multi user to single user. 'kill -1 1' is used to send the hangup signal to all processes.

**FILES**

*/dev/console*, */etc/utmp*, */usr/adm/wtmp*, */etc/ttys*, */etc/rc*

**SEE ALSO**

*kill(1)*, *login(1)*, *sh(1)*, *ttys(5)*, *getty(8)*



The Bell Laboratories UNIX command *f77(1)* (see the *UPM Volume 1*) runs a compiler which closely implements the ANSI standard X3.9 - 1978, known as FORTRAN 77. For a language reference document, see the ICL publication:

### *FORTRAN 77 Language*

which documents the language as implemented by the ICL Range compiler. Any slight differences between the two implementations are described in section A4.1. This appendix describes:

<i>Section</i>	<i>Contents</i>
A4.1	Language variations and implementation characteristics
A4.2	Mixed language programming
A4.3	Running a UNIX FORTRAN 77 program

#### **A4.1 Language variations and implementation characteristics**

UNIX FORTRAN 77 adheres closely to the ANSI FORTRAN 77 standard as defined in ANSI X3.9-1978. However, a few extensions to the language are provided to assist compatibility with other FORTRAN dialects, to allow specific features of UNIX on PERQ to be exploited or to facilitate communication with C procedures of the compiler, as detailed in the following subsections:

<i>Section</i>	<i>Contents</i>
A4.1.1	Language extensions
A4.1.2	Violations of the standard
A4.1.3	Implementation characteristics
A4.1.4	Source language effects

##### **A4.1.1 Language extensions**

Language extensions are allowable within the ANSI 77 standard since they do not conflict with standard definitions, but should not be used in programs that are intended to be portable to other implementations of FORTRAN 77. The use of FORTRAN 66 extensions are flagged with a warning by the compiler (see section A4.3.2.3); such warnings may be inhibited by use of a compiler option (see *Suppression of warnings*, section A4.3.2.2).

###### **A4.1.1.1 Lower case**

For consistency with UNIX system usage, lower case letters are expected in a FORTRAN source program, and upper case letters (except character literals) are transformed to lower case by default at compilation time. A compiler option (-U) can be used to inhibit this conversion (see *Non-standard language features*, section A4.3.2.2), but care must be taken in using this option as keywords will not be recognised if they are not in lower case.

Note: For consistency and easy recognition, FORTRAN statements are written in capital letters in the following sections.

###### **A4.1.1.2 Hollerith constants**

Hollerith data may be used in place of character string constants or to initialise non-character variables in DATA statements.

#### A4.1.1.3 EQUIVALENCE statements

An element of a multi-dimensional array in an EQUIVALENCE statement may be expressed as a singly-subscripted reference. The compiler will flag a warning for each missing subscript, and assume a value of one for each.

#### A4.1.1.4 One-trip DO loops

In the FORTRAN standard, a DO loop is not executed if the initial value of the DO loop variable is greater than the limit value. A compiler option in UNIX FORTRAN may be used to enable such loops to be executed once only (see *Non-standard language features*, section A4.3.2.2), for example:

**DO 10I = J, 4**    where J is 5

#### A4.1.1.5 Source inclusion

Source may be included from a file by using a FORTRAN statement of the form:

**INCLUDE filename**

INCLUDES may be nested to a depth of ten.

#### A4.1.1.6 Double complex data type

Each part of a complex variable is represented by a pair of double precision real variables. A *double complex* version of every complex built-in function is also provided. The specific function names begin with z instead of c.

#### A4.1.1.7 Internal files

UNIX FORTRAN allows internal files to be used in direct and unformatted I/O statements, in addition to the formatted sequential I/O statements to which they are restricted in the FORTRAN standard.

#### A4.1.1.8 IMPLICIT UNDEFINED statement

The FORTRAN standard has an IMPLICIT statement for overriding the traditional defined declaration rules (that is, a variable whose name begins with i, j, k, l, m or n is of type integer, other variables are of type real unless otherwise declared).

UNIX FORTRAN supports an additional type, the IMPLICIT UNDEFINED statement of the form:

**IMPLICIT UNDEFINED (a-z)**

The statement switches off the automatic typing mechanism, and the compiler issues a diagnostic for each variable that is used but which does not appear in a type statement. For example

**IMPLICIT UNDEFINED (i-n)**

overrides the automatic integer typing for variables beginning with the letters i, j, k, l and m.

Alternatively, a compiler option (-u) may be specified (see *Non-standard language features*, section A4.3.2.2), which is equivalent to beginning each program unit with the IMPLICIT UNDEFINED statement.

#### A4.1.1.9 Recursion

Direct and indirect recursion are permitted, that is procedures may call themselves directly, or indirectly through a chain of other procedures.

#### A4.1.1.10 Automatic storage

Two new keywords are recognised, *static* and *automatic*. These may appear as types in TYPE statements and IMPLICIT statements. Local variables are static by default; there is exactly one copy of the datum, and its value is retained between calls. For each variable declared automatic, there is one copy for each invocation of the procedure. Variables declared as automatic may not appear in EQUIVALENCE, DATA or SAVE statements.

#### A4.1.11 *Source input format*

The FORTRAN standard has a 72 position fixed format for input to the compiler as follows:

<i>Position</i>	<i>Use</i>
1 - 5	label (statement number)
6	continuation character
7 - 72	statement (padded with blanks if not used)
73+	comment (ignored by compiler)

The UNIX FORTRAN compiler has the additional facility to accept input of variable length lines, thus making typing of programs easier. This is achieved by:

- 1 The use of tab characters in one of the first six positions of a line, which signals the end of the statement number and continuation character. The renaming characters are treated as statement. (A tab elsewhere on the line is treated as a blank)
- 2 An ampersand (&) in the first position of a line indicates a continuation line; the remaining characters form the statement

#### A4.1.12 *Binary initialisation constants*

Variables of the types logical, integer or real can be initialised in a DATA statement by binary constants, denoted by a letter followed by a string in quotes where

b = binary (digits 0, 1)

o = octal (digits 9 to 7)

z or x = hexadecimal (digits 0 to 9, a to f)

For example, the following statements have the listed values:

<i>Statement</i>	<i>Value</i>
b '0001'	1
o '12'	10
z 'F'	15

#### A4.1.13 *Character strings*

##### *Backslash escapes*

For compatibility with C usage, the following backslash escapes are recognised:

\n newline

\t tab

\b backspace

\f form feed

\0 null

\' apostrophe (does not terminate a string)

\\" quotation mark (does not terminate a string)

\\\ \

\x x, where x is any other character

### *Quoting character*

UNIX FORTRAN permits double quotes ("), as well as the apostrophe ('), unlike the FORTRAN standard which has only the apostrophe. If a string begins with one variety of quote mark, the other may be embedded within it without using the repeated quote or backslash escapes.

### *Alignment*

Every unequivalenced scalar local character variable and every character string constant is aligned on an integer word boundary. Each character string constant appearing outside a DATA statement is followed by a null character to facilitate communication with C routines.

#### A4.1.1.14 *Commas in formatted input*

UNIX FORTRAN allows commas to be used as value separators in the input record for a formatted read of non-character variables. The commas override the field lengths given in the format statement. Thus, the format

(i10, f20.10, i4)

will read the record

-345, 05e -3, 12

correctly.

#### A4.1.1.15 *Short integers*

f77 accepts declarations of the type integer\*2 to allow halfword integers (C type short int) to be defined. A compiler option (-I2 flag) makes the default integer constants and variables short, and all quantities of the type logical will be short. An expression involving only objects of type integer\*2 is of short integer type.

If the precision of an integer valued intrinsic function is not determined by the generic function rules, one will be chosen that returns the prevailing length (integer\*2 when the -I2 option is specified).

Ordinary integers follow the FORTRAN standard rules about occupying the same space as a REAL variable and are assumed to be of C type, long int.

Note: Short integer and logical quantities do not obey the standard rules for storage association.

#### A4.1.1.16 *Additional intrinsic functions*

In addition to the intrinsic functions specified in the FORTRAN standard the compiler supports functions for performing bitwise Boolean operations (OR, AND, XOR and NOT) and for accessing the UNIX command arguments (GETARG and IARGC).

### A4.1.2 *Violations of the standard*

UNIX FORTRAN 77 violates the ANSI FORTRAN standard as described in the following sections.

#### A4.1.2.1 *Alignment of data*

The compiler imposes the rule that 32-bit quantities must be aligned on 32-bit boundaries, so care must be taken not to contravene this rule when using integer\*2 items (see section A4.1.1.15).

#### A4.1.2.2 *Dummy procedure arguments*

If a procedure contains any arguments of type CHARACTER, all dummy procedure arguments in that procedure must be declared in an EXTERNAL statement, otherwise a warning will be printed and incorrect object code may be generated.

#### A4.1.2.3 *T* and *TL* formats

The implementation of absolute and leftward tabbing with the use of T and TL formats is defective, in that it will not work with units which are connected to slow devices. When used with slow devices, for example a terminal, a failure will occur during execution of the FORTRAN program.

#### A4.1.2.4 *Unformatted direct access*

Unformatted direct access with a record length of 1, does not obey the rule of one record per direct access READ or WRITE statement. The file is accessed from the specified record number until the input/output list is exhausted.

#### A4.1.2.5 *A* formats

The use of the A format specifier to read character data into non-character variables causes data to be held in byte swapped format. In particular an A format read into variable I assigns characters to the bytes making up I in the order byte 1, byte 0, byte 3, byte 2. For example:

I = [byte 0|byte 1|byte 2|byte 3]

The following call on data 'ABC'

```
INTEGER I
READ (5,100) I
100 FORMAT (A3)
```

gives:

I = [B|A|-C]

#### A4.1.3 Implementation characteristics

##### A4.1.3.1 Storage of constants and variables

The size of constants and variables with their corresponding store and parameter alignments are given in Table A4.1.

**Table A4.1**  
Size of variables

Data type	Size	Store alignment	Parameter alignment
Real	4 bytes	4 bytes	4 bytes
Double precision	8 bytes	4 bytes	4 bytes
Logical	4 bytes	4 bytes	4 bytes
Character	1 byte	2 bytes	4 bytes
Integer	4 bytes	4 bytes	4 bytes
Integer*2	2 bytes	2 bytes	4 bytes

The maximum size of character variables is 32767.

The internal representation, range and precision of floating-point values conforms with IEEE standards (see *An Implementation Guide to a Proposed Standard on Floating Point*, Computer, January 1980).

### A4.1.3.2 Run-time file access

#### *File connections*

The default unit input file is pre-connected to the keyboard and the default unit output file is pre-connected to the display for serial formatted input/output. Unit 5 is also pre-connected to the keyboard for input and unit 6 to the display for output, and unit 0 for the error stream.

All other units are pre-connected for sequential formatted input/output, which may be overridden as follows:

- 1 By an OPEN statement, which allows a particular filename and any necessary file properties to be associated with a unit number
- 2 If no OPEN statement is obeyed before a unit is accessed, then the unit is opened implicitly with file properties appropriate to the input/output statement which first accesses the file

If no filename is associated with a unit number (by means of an OPEN statement), the unit is connected to a file in the current directory which has filename FORT.*n*, where *n* is the unit number.

#### *File types*

Unformatted sequential access files in FORTRAN are not compatible with unformatted direct access files.

#### *Compatibility of FORTRAN files with other languages*

FORTRAN files may be copied and listed using UNIX file utilities, but cannot be modified using the UNIX EDITOR.

#### *Opening files*

The following rules apply:

- 1 When a file is opened for sequential access, the FORTRAN system positions at the end of the file
- 2 An OPEN statement for an existing file, but with STATUS=NEW, does not fail. When connecting the file for sequential access the effect is first to empty the file
- 3 A connected file will be automatically closed when an OPEN statement, which associates a different file with the same unit number, is executed
- 4 STATUS=UNKNOWN is treated like STATUS=NEW if the file does not exist, otherwise it is interpreted in the same way as STATUS=OLD
- 5 When an OPEN statement is not used to open a file, pre-connection is to a file named FORT.*n* in the current directory, where *n* is the unit number
- 6 A non-existent file will be created when first accessed if it is not connected by execution of an OPEN statement

#### *Input/output*

- 1 On serial formatted input, if the input list exceeds the record size no error is reported but the record is extended with blanks
- 2 On direct access input where the record number is beyond the file, the error message  
**eof/uio: Not a typewriter**  
is reported
- 3 Formatted data transfer between incompatible data types will not always fail; the value will be converted according to the edit descriptor, where possible
- 4 If there is no repeatable edit descriptor corresponding to an input/output list item, the statement will not always fail. A non-repeatable edit descriptor value (for example, a character string) may be transferred

- 5 Sequential access beyond an end-of-file record is not reported as being an invalid access. For example, the sequence

**WRITE...ENDFILE...WRITE**

will be implemented as if there were no ENDFILE statement

- 6 The I/O system appends a new-line character to each formatted record output. On input, a new-line character is used as a record separator

*Unit numbers*

Unit numbers must be in the range 0-19 inclusive; this is a constraint imposed by the I/O library.

#### A4.1.3.3 *Unnamed block data*

More than one unnamed block data subprogram is allowed in an executable program.

#### A4.1.4 Source language effects

##### A4.1.4.1 *PAUSE statement*

When a FORTRAN PAUSE<*ident*> statement is executed, where <*ident*> is an integer or a text string or null, the message:

**PAUSE<ident>** statement executed. To resume execution, type **go**. Any other input will terminate job.

is sent to the screen. If the user types **go** in reply, the program displays the message:

**execution resumes after PAUSE**

and execution continues.

If the user types any other reply, the program displays the message:

**STOP**

and terminates.

##### A4.1.4.2 *STOP statement*

When a FORTRAN STOP<*ident*> statement is executed, where <*ident*> is an integer or a text string, the message:

**STOP<ident>** statement executed

is displayed on the screen and the program is terminated.

A message is displayed only if the STOP statement contains a message (string or integer).

#### A4.1.5 *Intrinsics*

The intrinsics available are as described in the ICL publication *FORTRAN 77 Language*, plus the following:

*Intrinsic*      *Definition*

**DCMPLX**      As CMPLX but type of function is DOUBLE COMPLEX

**DFLOAT**      As FLOAT but type of function is DOUBLE PRECISION

**AND**

**OR**

**XOR**

**NOT**

**LSHIFT**

**RSHIFT**

<b>ZABS</b>	As CABS but argument is DOUBLE COMPLEX and type is DOUBLE PRECISION
<b>IMAG</b>	A new generic name covering AIMAG and DIMAG
<b>DIMAG</b>	As AIMAG but argument and type as ZABS
<b>DCONJG</b>	As CONJG except argument and type as ZABS
<b>ZSQRT</b>	As CSQRT but argument and type are DOUBLE COMPLEX
<b>ZEXP</b>	As CEXP but argument and type are DOUBLE COMPLEX
<b>ZLOG</b>	As CLOG but argument and type are DOUBLE COMPLEX
<b>ZSIN</b>	As CSIN but argument and type are DOUBLE COMPLEX
<b>ZCOS</b>	As CCOS but argument and type are DOUBLE COMPLEX

Note: The *f77* compiler explicitly converts all parameters being passed to the directly called intrinsics (SIN, COS, TAN, SQRT, EXP, LOG, ASIN, ACOS, ATAN, ATAN2, SINH, COSH, TANH) from type FLOAT to type DOUBLE PRECISION.

#### A4.2 Mixed language programming

Mixed language programming with C language is supported to allow users access to system procedures and PERQ facilities not directly available from FORTRAN.

A UNIX FORTRAN object program can call C procedures, and vice versa, and it will be possible to share global data. C procedures that call, or are called by, FORTRAN procedures can be written provided that the conventions for procedure names, data representations, return values and argument lists are obeyed.

##### A4.2.1 Procedure names

On UNIX systems, the compiler appends an underscore to the name of a common block or a FORTRAN procedure, to distinguish it from a C procedure or external variable with the same user-assigned names. FORTRAN library procedure names have embedded underscores to avoid clashes with user-assigned subroutine names.

##### A4.2.2 Data representation

Table A4.2 defines the corresponding FORTRAN and C declarations.

**Table A4.2**  
FORTRAN and C declarations

FORTRAN	C
integer*2 x	short int x;
integer x	int x;
integer x	long int x;
logical x	int x;
logical x	long int x;
real x	float x;
double precision x	double x;
complex x	struct { float r, i; } x;
double complex x	struct { double dr, di; } x;
character*6 x	char x[6];

Note: By the rules of FORTRAN, integer, logical and real data occupy the same amount of memory.

#### A4.2.3 Return values

A function of type integer, logical, real or double precision is declared as a C function that returns the corresponding type. A complex or double complex function is equivalent to a C routine with an additional initial argument that points to the place where the return value is to be stored. For example:

```
complex function f(...)  
is equivalent to  
f_(temp...)  
struct {float r, i;} *temp;  
...
```

A character-valued function is equivalent to a C routine with two extra initial arguments, that is a data address and a length.

For example:

```
character*15 function g(...)  
is equivalent to  
g_(result, length...)  
char result[];  
int length;  
...  
and could be invoked by C by  
char chars[15];  
...  
g_(chars, 15L,...);
```

Subroutines are invoked as if they were integer-valued functions whose value specifies which alternate return to use. Alternate return arguments (statement labels) are not passed to the function, but are used to do an indexed branch in the calling procedure.

Note: If the subroutine has no entry points with alternate return arguments, the returned value is undefined. The statement

```
call nret (*1, *2, *3)  
is treated exactly as if it were the computed goto  
goto(1, 2, 3), nret()
```

Note: If a FORTRAN program is to use the returned value of a C or Pascal function, and the returned value is a FLOAT or REAL in C or Pascal terms, then the function must be of type DOUBLE PRECISION within the FORTRAN program because the C and Pascal compilers insert on returning a DOUBLE. For example:

<i>FORTRAN</i>	<i>C</i>
EXTERNAL CFUNC	float CFUNC (Zparm);
DOUBLE PRECISION CFUNC	float Zparm;
.	.
Y = X + CFUNC(Z)	{
.	.
.	return (...)
.	}

#### A4.2.4 Argument lists

All FORTRAN arguments are passed by address. In addition, for every argument that is of type character or that is a dummy procedure, an argument giving the length of the value is passed. (The string lengths are long int quantities passed by value). The order of arguments is then:

- 1 Extra arguments for complex and character functions
- 2 Address for each datum or function
- 3 A long int for each character or procedure argument

For example, the call in

```
external f
character*7 s
integer b(3)
...
call sam(f, b(2), s)
```

is equivalent to that in

```
int f();
char s[7];
int b[3];
...
sam_(f, &b[1], s, OL, 7L);
```

Note that the first element of a C array always has subscript zero, but FORTRAN arrays begin at 1 by default. FORTRAN arrays are stored in column-major order. C arrays are stored in row-major order.

### A4.3 Running a UNIX FORTRAN 77 program

This chapter explains how a UNIX FORTRAN 77 source program is converted into an object (runnable) program, and describes the utilities required to prepare the source program, to perform the conversion (source program to object program) and to run the object program.

Note: FORTRAN source files are transportable to other UNIX systems. However, the FORTRAN compilation and execution systems, and FORTRAN object programs produced by this compiler, are not practically portable because of differences in object file formats.

#### A4.3.1 Introduction

The main components of the compilation system are:

- 1 An options, syntax and semantic analyser written in C, which produces intermediate code (see section A4.3.2)
- 2 A run time library to support FORTRAN input/output, written in C
- 3 A run time library to support the intrinsic functions

##### A4.3.1.1 FORTRAN program conversion

A UNIX FORTRAN source program is held in a file which by convention has an extension .f (for example, *filename.f*).

When FORTRAN source files are compiled, an assembly source file with an extension of .s and an object file with an extension of .o are produced for each named source file. The object files are consolidated into a single run file called **a.out**. Assembly and object files are given the same name as the corresponding FORTRAN source file with the extension .s or .o respectively. At the end of assembly, the assembly source files disappear.

Assembly, object and run files are created if they do not exist, or are overwritten if they do exist.

##### A4.3.1.2 Utilities

The utilities used to prepare, convert and run a UNIX FORTRAN 77 program are:

- 1 *ed(1)* or *spy(1)*, *ed(1)* is the UNIX editor which is used initially to input, and later to amend, the source program. The editor is described in *UNIX Programmer's Manual*. *spy(1)* is the PNX interactive screen based text editor
- 2 *f77(1)* This calls several compilation phases (see Figure A4.1). It takes FORTRAN source files and produces a runnable program in the following phases:
  - (a) PREPROCESSOR *f77* calls the preprocessor which processes INCLUDE statements, and produces intermediate code.
  - (b) CODE GENERATOR This converts the intermediate code to assembly code, with an option for object code optimisation

- (c) ASSEMBLER This transforms assembly code to C image format
- (d) LINKER/LOADER (CONSOLIDATOR) This produces a relocatable binary file from the C image file and loads it. This is a mandatory process. This phase can also be performed using the *ld* command (see section A4.3.3)

Once a program has been successfully loaded, it can be run by issuing its filename as a command to the shell.

The compilation system also includes a low level input/output access library which is used by the run time library (standard C I/O package).

#### A4.3.2 Compilation

The utility f77 compiles a UNIX FORTRAN 77 program (.f file) to produce an assembly source file (.s file) and an object file (.o file) for each named source file. The object files are consolidated into a single executable file called a.out.

Note: The compiler does not produce any listings.

##### A4.3.2.1 *Using the compiler*

The compiler is invoked by issuing the command line:

**f77 [<option>]...<file>...**

where

<file> is a mandatory parameter which identifies the name of an input source file. FORTRAN source files must have the extension .f.

<option> can be used to specify the following:

- 1 If no options appear, all named source files are compiled by default (see section A4.3.1.1)
- 2 Options may be used to override the default compilation actions (see section A4.3.2.2)
- 3 The options may include loader options to enable previously compiled object files to be consolidated into the runnable file produced by f77. Alternatively, the loader may be invoked separately to consolidate object files (see section A4.3.3)

Note: Unrecognised compiler options are passed through to the consolidation phase, where any invalid options will be flagged by the loader.

The compiler sends the following standard title message to the screen:

<source-file-name>:

  <type> <program-name>:

where <type> is MAIN, SUBROUTINE or FUNCTION, and this line is repeated for each FORTRAN section compiled.

Note: It is not possible to call FORTRAN SUBROUTINES or FUNCTIONS direct from Shell (see section A4.3.4).

If the main program name is omitted from the source, or if the source file contains no main program, the following three lines are appended:

```
undefined
-MAIN---
-end
```

A similar message is appended for any procedures which are referenced from, but not declared in, the source file with MAIN replaced by the name of the procedure as follows:

```
undefined
-procedure name---
-end
```

Procedures must be linked in prior to execution (see section A4.3.3).

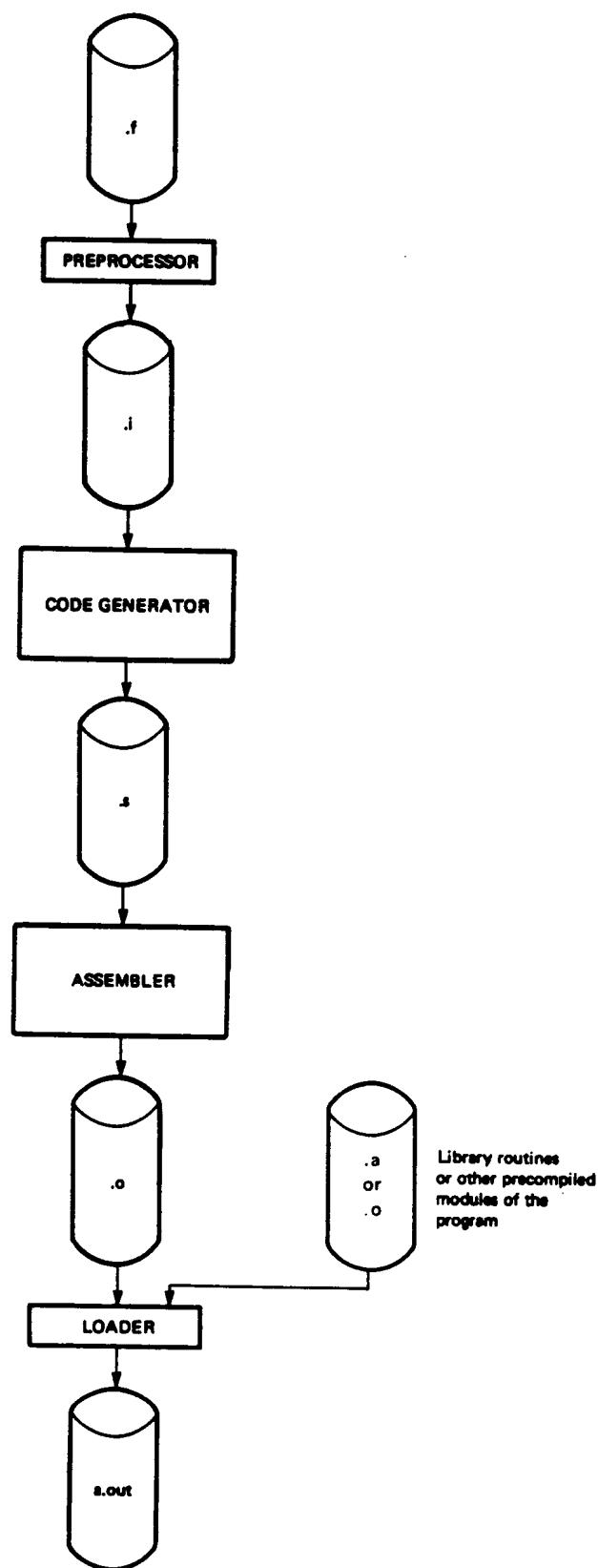


Figure A4.1 f77 compilation phases

### *Non-FORTRAN compilation*

*f77* may also be used to compile programs written in Ratfor and C, and to consolidate them with programs written in UNIX FORTRAN 77. These languages are described in the *UPM*.

#### A4.3.2.2 *Compiler options*

##### *Inhibit load*

The normal compilation process follows several stages:

- 1 Compilation into assembly source files (.s)
- 2 Assembly of source files into object files (.o)
- 3 Linking, which transforms the object file into relocatable binary format
- 4 Loading, which loads the linked object files into a single runnable file (a.out)

The process can be stopped using the following option at the defined stage:

- 1 **-c** To suppress the loading of object files (.o)

##### *Private run files*

By default, the runnable file is directed to a.out. The user may specify a different file using the option

**-o filename**

which loads the runnable file into *filename* instead of a.out. If the file does not exist, it will be created; if it already exists, it will be overwritten.

##### *Suppression of warnings*

Warnings normally output by the compiler may be inhibited using the options:

- 1 **-w** To suppress all warning messages
- 2 **-w66** To suppress warnings about FORTRAN 66 features used

##### *Diagnostic options*

The compiler will generate code to check that, at run time, subscripts are written within array bounds using the option

**-C**

The code checks only that the evaluated subscript lies within the array bounds. It does not check whether individual dimension bounds are exceeded.

Note: This option is always performed during compilation if the subscript is a constant expression.

##### *Object program performance*

The following options are available:

- 1 **-O** To generate optimised object code
- 2 **-p** To generate code to produce usage profiles (see section A4.3.5.2)

##### *Non-standard language features*

The following options are available to utilise certain non-standard features of UNIX FORTRAN 77 (see also sections A4.1.1.1, A4.1.1.4, A4.1.1.8 and A4.1.1.15).

- 1 **-onetrip** To perform DO loops at least once, when the upper limit is less than the lower limit
- 2 **-u** To make the default type of a variable UNDEFINED rather than using the default FORTRAN rules
- 3 **-U** To retain upper case letters; by default FORTRAN programs are converted to lower case
- 4 **-I2** To make default integer and logical values short, that is 16-bits long

*Non-FORTRAN compilation*

f77 may be used to compile Ratfor, in which case the following options may be used:

- 1 **-F** Preprocess Ratfor source files into FORTRAN (.f) source files, which are retained but not compiled
- 2 **-R** Use the remaining characters in the argument as a Ratfor flag argument

*Compiler overflow*

Whenever one of the compiler's fixed length tables overflows the compiler issues the appropriate message below:

Too many loops or if-then. elses.	Try the -Nc option.
Too many equivalents.	Try the -Nq option.
Too many names.	Try the -Nn option.
Too many statement numbers.	Try the -Ns option.
Too many external symbols.	Try the -Nx option.

c, q, n, s, or x refer to the particular table. Their default sizes are as follows:

c 20  
q 150  
n 401  
s 201  
x 200

To overcome the problem, recompile with the -N option. The option has the form:

**-N?d**

where ? is one c, q, n, s or x and d is a decimal number specifying the new size of the appropriate table.

#### A4.3.2.3 *Compile time diagnostics*

Two categories of message are produced at compile time:

- 1 **WARNINGS** For example, use of a non-standard feature
- 2 **ERRORS** For example, syntax error

Diagnostic messages are output to the display in the form:

{Warning } on line n of source-file-name: explanation...  
{Error }

Note that these warnings can be inhibited by a compiler option.

If errors are detected during compilation, no assembly source file is produced or assembled, in which case the line:

Error. No assembly

is appended to the compilation messages.

If warnings only are detected during compilation, then compilation continues through assembly and consolidation.

In addition, errors may be reported from the operating system when, for example, file or main store limits are exceeded.

#### A4.3.3 Consolidation

The consolidation process combines the link/load functions which ensure that all referenced procedures are linked together in the final object program. This can be achieved in either of the following ways:

- 1 By use of the explicit loader options supplied to f77 (see section A4.3.2.1)
- 2 By use of a specific loader command of the form:

**ld [<option>...]<file>...**

This command is described in detail in the *UNIX Programmer's Manual*

Use of *f77* will cause each of the procedures compiled by the command to be automatically linked. All other user programs to be consolidated need to be named in the *f77* or *ld* command. Since they have the suffix *.o*, they are linked directly.

If any standard library procedures are used, these must also be consolidated. The *f77* command automatically links in all referenced FORTRAN library procedures; for other standard libraries, or if the *ld* command is used, relevant libraries must be nominated with the *-l* option.

For standard FORTRAN library procedures, the names, in order, are F77, I77 and C. The order is extremely important because the linker searches the named libraries in order and will not satisfy external references to procedures in a library which has already been processed (see *UNIX Programmer's Manual* for details).

The loadable program will be entered through its main program entry point.

#### A4.3.4 Running the program

FORTRAN programs are entered through the main program unit by a command line which names the file containing the main program. This will be either:

- 1 **a.out**
- 2 The name specified in an option to *f77* or *ld*

It is not possible to call SUBROUTINES or FUNCTIONS directly from the shell, since entry must initially be from a main program.

It is possible to invoke FORTRAN SUBROUTINES and FUNCTIONS from programs compiled in other languages (see section A4.2).

##### A4.3.4.1 Run time diagnostics

The following categories of error can occur at run time:

- 1 **HARDWARE DETECTED** For example, zero divide, bound check, overflow
- 2 **ERRORS IN INTRINSIC FUNCTIONS**
- 3 **INPUT/OUTPUT**
- 4 **SOFTWARE DETECTED**

On detection of an error, execution is halted and the FORTRAN 77 diagnostic procedure is entered. This outputs an error message and information relating to the FORTRAN statement currently being executed. The program then dumps to a file **core** in the current directory, and terminates.

For input/output errors, the diagnostic output has the following form:

```
<error-code> : <error-message>
apparent state : unit <n> named <filename>
last format : <format>
lately <I/O action>
IOT trap - core dumped
```

where

<error-code> is a cryptic error type, for example, eof/uio  
(end-of-file or unrecoverable I/O error)

<error-message>, for example, formatted I/O not allowed

<n> is the unit number

<filename> is the name of the file

<format> is the source format (blank for unformatted I/O)

<I/O action> describes the type of input/output action attempted, for example, writing sequential formatted external I/O

For other errors, the diagnostic output is of the form:

<error-message> on file line <n>, procedure <section-name>

<additional-info>

IOT trap - core dumped

where

<error-message> indicates the type of error, for example, subscript out of range

<n> is the line number on which the error occurred

<section-name> is the section containing this line

<additional-info> gives more specific information, for example, attempt to access the 70th element of variable *array*

Except for the program dump to file core, all error information is output to the screen. The file core is not in printable form, but may be interrogated by use of the UNIX Debugger *sdb(1)* (see Chapter 8).

#### A4.3.5.2 Run time profiles

Information on the time spent in the various procedures during execution of a FORTRAN program can be obtained by using the *prof* command (see *UNIX Programmer's Manual*). The procedures must have been compiled with the *-p* option set.

The information can be presented in tabular or graphic form.

### A5.1 Language variations and implementation characteristics

#### A5.1.1 Language extensions

Since the C language was described in *The C Programming Language* (Kernighan and Ritchie), the authors have designed and implemented two extensions; the first permits structure assignment, and the second provides an extra type: the enumeration type. These two extensions are described in *Recent Changes to C*, an addendum to section 14 of the UPM, volume 2.

The C programming language has been implemented under PNX on the ICL PERQ just as described in the publication *The C Programming Language* together with the two extensions just mentioned. There are no omissions, and no further extensions.

Where PERQ-oriented features are required in C programs, they are invoked by the use of special functions, rather than by extensions to the C language.

#### A5.1.2 Implementation characteristics

The following table shows, for each type of single variable, the number of bits of information, and the alignment of the variable in main store and when passed as a parameter:

Type	Information content	Store alignment	Parameter alignment
char	8	16	32
int	32	32	32
short	16	16	32
long	32	32	32
float	64	32	32
double	64	32	32

The ICL PERQ uses the ASCII character code (see Appendix 6). A single-character *char* variable is stored in the least significant 8 bits of one 16-bit word. However, arrays of type *char* are stored with two characters to each word, except the last, if there is an odd number of characters, in which case it is stored in the least significant 8 bits of the last 16-bit word used to hold the array.

Variables of type *int*, *short* and *long* all use the most significant bit as the sign bit, and are stored in twos-complement form.

The representation of variables of type *float* and *double* conform to the IEEE standard for floating point (see *A Proposed Standard for Floating-Point Arithmetic*, Computer, March 1981)

### A5.2 Mixed language programming

#### A5.2.1 UNIX FORTRAN 77

C programs can call UNIX FORTRAN 77 procedures and UNIX FORTRAN 77 programs can call C procedures. Global data can be shared. Details are given in A4.2.

### A5.3 Running a C program

#### A5.3.1 Inputting, correcting and examining a source program

You can use *ed* or *spy* to:

- 1 Input a new program. Remember to give a C source program an extension of .c
- 2 Correct an existing program
- 3 Examine all or part of a source program file

For details of the UNIX text editor command, see *ed(1)* in volume 1 of the UPM. See also *A Tutorial Introduction to the UNIX Text Editor* in section 4, and Advanced Editing on UNIX in section 5 of volume 2 of the UPM.

#### A5.3.2 Compiling a program

You can initiate a compilation by issuing the command *cc*, see *cc(1)* in Volume 1 of the UNIX Programmer's Manual. On ICL PERQ the UNIX command *pcc* calls the same compiler as *cc*, namely the portable C compiler, thus there is no advantage in using *pcc*.

Compilation proceeds in several phases:

- 1 Preprocessing
- 2 Code generation and assembly
- 3 Loading

as illustrated in Figure A5.1.

##### A5.3.2.1 Preprocessing

The .c file may contain lines starting # (see section 12 in *The C Programming Language - Reference Manual*, which is section 14 of the UPM, Volume 2). The preprocessor phase reads the .c file and takes appropriate actions as directed by these lines, and writes the output to a .i (intermediate) file which no longer contains such lines. The *cc* command can be terminated after this phase by the -P option. The preprocessor is controlled by the # line syntax, which is not related to the C language syntax. Therefore the preprocessor can be used on any textual file, if you so wish.

The preprocessor phase is automatically performed on a .c file even though it may contain no # lines.

##### A5.3.2.2 Code generation and assembly

The code generation phase reads the .i file and writes the generated code to a .s (assembler) file. The code at this stage is in mnemonics of an assembly language unique to the ICL PERQ C-machine. The assembly phase then reads the .s file and writes the program to a .o (object) file. The code at this stage is in a relocatable form of C-code, the form in which library routines are held. The *cc* command can be terminated after this phase by the -c option.

##### A5.3.2.3 Loading

UNIX terminology is somewhat confusing at this point, since many people take the term *loading* to mean the production of a program image in main store immediately prior to execution. In UNIX however, loading refers to the process of creating an absolute binary image on file, rather than in main store. The loading phase is performed for *cc* by the *ld* command (see *ld(1)* in volume 1 of the UNIX Programmer's Manual). It reads the .o file, together with any other relocatable object files (.o), such as library routines and other precompiled parts of the program, and writes an absolute binary image of the complete program to the a.out file by default, or to the -o option file if given.

#### A5.3.3 Running a program

To run a program which has successfully compiled, issue the name of the file output from the loading stage (that is, either a.out or the parameter supplied to the -o option) as a command to the shell.

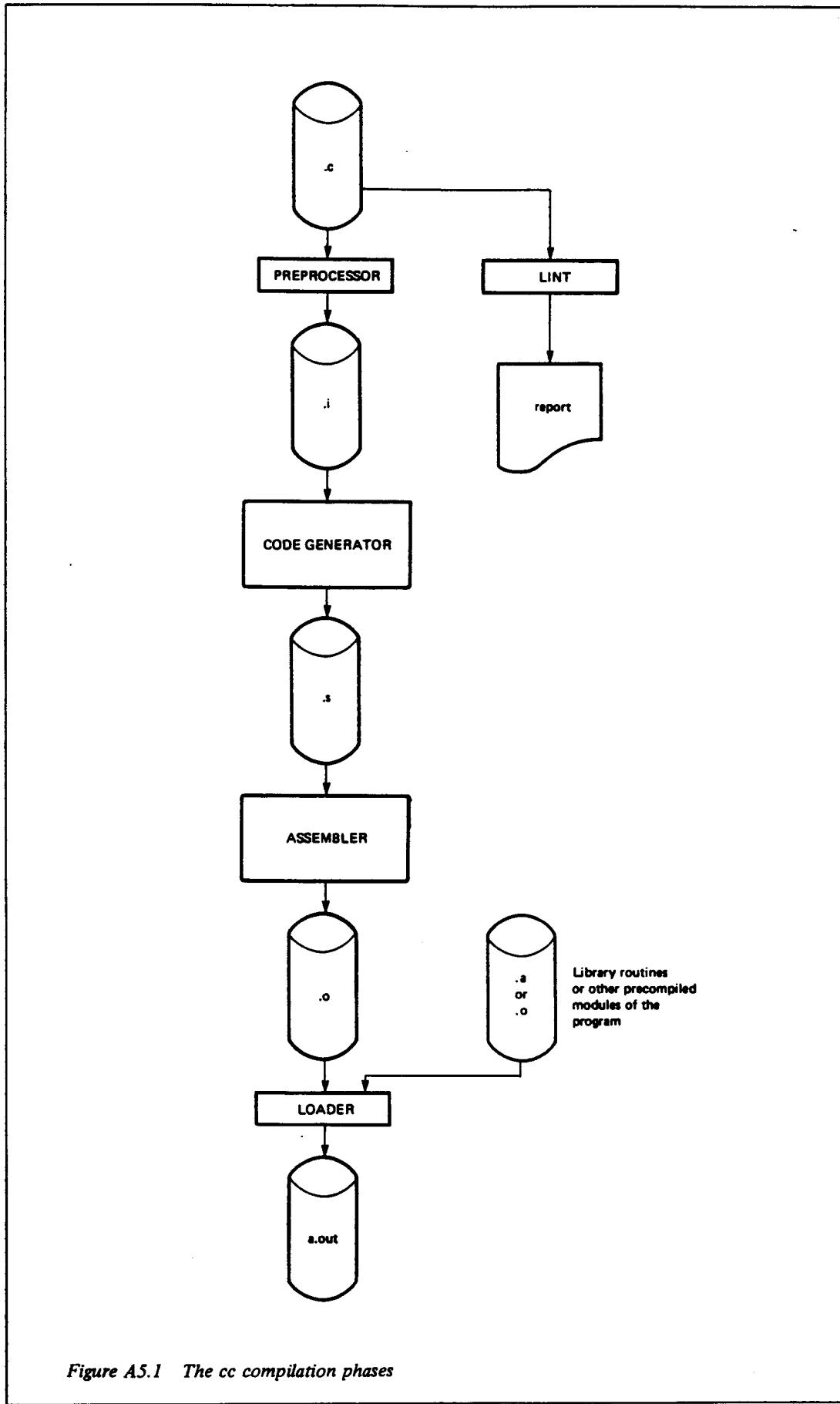


Figure A5.1 The cc compilation phases

### A5.3.4 Debugging

PNX contains the *lint* debugging tool. *Lint* is a C program checker. Whereas the compiler concentrates on quickly and accurately producing a runnable program from the program text, *lint* makes comments on its portability, style, and efficiency. *Lint*, since it is not generating a program, merely commenting on it, can risk being wrong, since it is up to the programmer to interpret its output.

*Lint's* comments, for example on variables declared but never used, or code which can never be reached, can help to spot bugs, particularly in long and heavily amended programs.

For details, see *Lint(1)* in the UPM Volume 1, and *Lint, a C Program Checker*, UPM Volume 2 section 15.

## A5.4 Hints on porting C programs for PNX

### A5.4.1 C-machine sizes

The sizes of various C object under PNX are:

Type	Length in bits
char	8
int	32
short	16
long	32
float	64
double	64

Floating point numbers conform to the IEEE standard: 1 sign bit, and 11 bit excess 1023 exponent (used as a power of 2) and a 52 bit normalised mantissa.

### A5.4.2 Using lint

*Lint* is used to check programs for portability. Unfortunately, it is only of limited use since many C programs break the type rules quite harmlessly. One of the few options worth using in this context is -c. Much of the output can be disregarded. PNX uses 32 bits to hold a pointer, so any message reading:

“illegal combination of pointer and integer”

can be ignored.

The -lc option is useful to allow *lint* to see the declarations of the standard C library routines. *Lint* may then check the types of the arguments you pass to them.

The most serious error messages are:

“illegal combination of pointers”

“argument <num> of <function> used inconsistently”

“mixed mode of return used”

Do not get obsessed with *lint* output. Often the simplest way of correcting a porting bug is to add a *cast* (see *The C Programming Language*) which actually makes *lint* complain even more.

### A5.4.3 Pointers

Almost all porting difficulties arise because on PNX a piece of data may be pointed at in two different ways: byte pointers and word pointers.

On a word addressed machine like PERQ it is desirable to have a way of pointing to an arbitrary byte. The C-machine constructs such a byte pointer by shifting the address of the word containing the byte left one place and indicating the low or high order byte by a 0 or 1 in the bottom bit.

The C types *pointer to character(s)* and *array of characters* are represented by byte pointer. A pointer to any other type will be a word pointer.

For example, in:

```
struct one-piece {  
    ...  
    ...  
} lots-of-pieces[42];  
main()  
{  
    ...  
    ...  
    fread(&lots-of-pieces[21], sizeof(struct one-piece), 21, fp  
    ...  
    ...  
}
```

the intention is clearly to read 21 'structure fulls' of data from the stream *fp* into the elements 21 to 41 of the array lot-of-pieces. This is what would happen on a PDP or VAX for example. The expression '&lots-of-pieces[21]' is of type 'struct one-piece \*' and is therefore a word pointer. fread however is expecting a byte pointer so a cast is required. The line starting fread should be:

```
fread((char *)&lots-of-pieces[21], sizeof ...)
```

A more complex problem arises if the compiler has lost track of the type of pointer involved because it has been assigned to an integer. For example, using the same definitions as above:

```
main()  
{  
    int temp = &lots-of-pieces[21];  
    ...  
    ...  
    muddle(temp);  
    ...  
    ...  
}  
muddle(ptr)  
{  
    ...  
    ...  
    fread((char *)ptr, sizeof(struct one-piece), 21, fp); ....  
}
```

The expression '(char \*)ptr' has no effect, because the compiler does not know how to convert an integer to a pointer, so it does nothing. The code required is:

```
fread((char *)(struct one-piece *)ptr, sizeof...)
```

that is, first tell the compiler what type ptr really is, then tell it what it should be.

There are even more difficult problems involved if the error is hidden by a level of indirection. Logically, the cast '(char \*\*)x' when applied to an array of type 'struct one-piece \*\*' should convert every ELEMENT in the array to type 'char \*' but this is beyond the ability of most compilers. Such cases are fortunately quite rare.



On the ICL PERQ, a character is often stored in a single byte, the most significant bit of which is bit 0 and the least is bit 7. Since ASCII is a seven-bit code, only bits 1 to 7 are used for the actual ASCII code.

The ASCII set comprises four main parts:

- 1 **CONTROL** (bit 1=0, bit 2=0) These 32 codes are almost never meant to be displayed; instead they cause software to take some action. For example, code 13 (octal 15) is CR (carriage return) and is generated from the keyboard by pressing RETURN
- 2 **SYMBOLS** (bit 1=0, bit 2=1) The digits 0 to 9 together with 22 common symbols such as +, - and .
- 3 **UPPERCASE LETTERS** (bit 1=1, bit 2=0) The 26 uppercase letters and 6 miscellaneous symbols
- 4 **LOWERCASE LETTERS** (bit 1=1, bit 2=1) The 26 lowercase letters, 5 miscellaneous symbols and the delete code (bits 1 to 7 all=1)

The raw key value does not necessarily correspond to the ASCII code and the PERQ keyboard provides more keys than the ASCII codes. Ordinarily, key presses are translated by *tty* (4) and non-ASCII codes have no effect. Raw key values are given so you can program the keyboard effects.

<i>Ordinal: Dec Oct</i>	<i>Bits: 123 4567</i>	<i>Code name</i>	<i>Keystroke(s)</i>	<i>Raw key value</i>	<i>Comments</i>
0 #0	000 0000	NUL	CTRL SHIFT @	1100 0000	Null
1 #1	000 0001	SOH	CTRL A	1110 0001	Start of heading
2 #2	000 0010	STX	CTRL B	1110 0010	Start of text
3 #3	000 0011	ETX	CTRL C	1110 0010	End of text
4 #4	000 0100	EOT	CTRL D	1110 1000	End of transmission
5 #5	000 0101	ENQ	CTRL E	1110 0101	Enquiry
6 #6	000 0110	ACK	CTRL F	1110 0110	Acknowledge
7 #7	000 0111	BEL	CTRL G or HELP	1110 0111	Bell
8 #10	000 1000	BS	CTRL H or BACKSPACE	1110 1000 0000 1000	Backspace
9 #11	000 1001	HT	CTRL I or TAB	1110 1001 0000 1001	Horizontal tabulation
10 #12	000 1010	LF	CTRL J or LF	1110 1010	Line feed
11 #13	000 1011	VT	CTRL K	1110 1011	Vertical tabulation
12 #14	000 1100	FF	CTRL L	1110 1100	Form feed
13 #15	000 1101	CR	CTRL M or RETURN	1110 1101 0000 1101	Carriage return
14 #16	000 1110	SO	CTRL N	1110 1110	Shift out
15 #17	000 1111	SI	CTRL O	1110 1111	Shift in
16 #20	001 0000	DLE	CTRL P	1111 0000	Data link escape
17 #21	001 0001	DC1	CTRL Q	1111 0001	Device control 1
18 #22	001 0010	DC2	CTRL R	1111 0010	Device control 2
19 #23	001 0011	DC3	CTRL S	1111 0011	Device control 3
20 #24	001 0100	DC4	CTRL T	1111 0100	Device control 4
21 #25	001 0101	NAK	CTRL U or OOPS	1111 0101 0001 0101	Negative acknowledge
22 #26	001 0110	SYN	CTRL V	1111 0110	Synchronous idle
23 #27	001 0111	ETB	CTRL W	1111 0111	End of transmission block
24 #30	001 1000	CAN	CTRL X	1111 1000	Cancel
25 #31	001 1001	EM	CTRL Y	1111 1001	End of medium
26 #32	001 1010	SUB	CTRL Z	1111 1010	Substitute
27 #33	001 1011	ESC	CTRL [ or INS	1111 1011	Escape
28 #34	001 1100	FS	CTRL \	1101 1100	File separator
29 #35	001 1101	GS	CTRL SHIFT ]	1101 1101	Group separator
30 #36	001 1110	RS	CTRL SHIFT ^	1101 1110	Record separator
31 #37	001 1111	US	CTRL SHIFT _	1101 1111	Unit separator
32 #40	010 0000	SP	Spacebar	0010 0000	Space
33 #41	010 0001	!	SHIFT !	0010 0001	Exclamation mark
34 #42	010 0010	"	SHIFT "	0010 0010	Quote
35 #43	010 0011	#	SHIFT #	0010 0011	Number sign (hash mark)
36 #44	010 0100	\$	SHIFT \$	0010 0100	Dollar sign
37 #45	010 0101	%	SHIFT %	0010 0101	Percent sign
38 #46	010 0110	&	SHIFT &	0010 0110	Ampersand
39 #47	010 0111	'	'	0010 0111	Apostrophe (same key as Quote)
40 #50	010 1000	(	SHIFT (	0010 1000	Left parenthesis
41 #51	010 1001	)	SHIFT )	0010 1001	Right parenthesis
42 #52	010 1010	*	SHIFT *	0010 1010	Asterisk
43 #53	010 1011	+	SHIFT +	0010 1011	Plus sign
44 #54	010 1100	,	,	0010 1100	Comma
45 #55	010 1101	-	-	0010 1101	Minus sign (hyphen)
46 #56	010 1110	.	.	0010 1110	Full stop
47 #57	010 1111	/	/	0010 1111	Oblique
48 #60	011 0000	0	0	0011 0000	
49 #61	011 0001	1	1	0011 0001	
50 #62	011 0010	2	2	0011 0010	
51 #63	011 0011	3	3	0011 0011	
52 #64	011 0100	4	4	0011 0100	
53 #65	011 0101	5	5	0011 0101	
54 #66	011 0110	6	6	0011 0110	
55 #67	011 0111	7	7	0011 0111	
56 #70	011 1000	8	8	0011 1000	
57 #71	011 1001	9	9	0011 1001	
58 #72	011 1010	:	SHIFT :	0011 1010	Colon
59 #73	011 1011	;	;	0011 1011	Semicolon
60 #74	011 1100	<	SHIFT <	0011 1100	Less than sign
61 #75	011 1101	=	=	0011 1101	Equals sign
62 #76	011 1110	>	SHIFT >	0011 1110	Greater than sign
63 #77	011 1111	?	SHIFT ?	0011 1111	Question mark

<i>Ordinal: Dec Oct</i>	<i>Bits: 123 4567</i>	<i>Code name</i>	<i>Keystroke(s)</i>	<i>Raw key value</i>	<i>Comments</i>
64 #100	100 0000	@	SHIFT @	0100 0000	
65 #101	100 0001	A	SHIFT A	0100 0001	
66 #102	100 0010	B	SHIFT B	0100 0010	
67 #103	100 0011	C	SHIFT C	0100 0011	
68 #104	100 0100	D	SHIFT D	0100 0100	
69 #105	100 0101	E	SHIFT E	0100 0101	
70 #106	100 0110	F	SHIFT F	0100 0110	
71 #107	100 0111	G	SHIFT G	0100 0111	
72 #110	100 1000	H	SHIFT H	0100 1000	
73 #111	100 1001	I	SHIFT I	0100 1001	
74 #112	100 1010	J	SHIFT J	0100 1010	
75 #113	100 1011	K	SHIFT K	0100 1011	
76 #114	100 1100	L	SHIFT L	0100 1100	
77 #115	100 1101	M	SHIFT M	0100 1101	
78 #116	100 1110	N	SHIFT N	0100 1110	
79 #117	100 1111	O	SHIFT O	0100 1111	
80 #120	101 0000	P	SHIFT P	0101 0000	
81 #121	101 0001	Q	SHIFT Q	0101 0001	
82 #122	101 0010	R	SHIFT R	0101 0010	
83 #123	101 0011	S	SHIFT S	0101 0011	
84 #124	101 0100	T	SHIFT T	0101 0100	
85 #125	101 0101	U	SHIFT U	0101 0101	
86 #126	101 0110	V	SHIFT V	0101 0110	
87 #127	101 0111	W	SHIFT W	0101 0111	
88 #130	101 1000	X	SHIFT X	0101 1000	
89 #131	101 1001	Y	SHIFT Y	0101 1001	
90 #132	101 1010	Z	SHIFT Z	0101 1010	
91 #133	101 1011	[	[	0101 1011	Left square bracket
92 #134	101 1100	\	\	0101 1100	Reverse oblique
93 #135	101 1101	]	SHIFT ]	0101 1101	Right square bracket
94 #136	101 1110	'	SHIFT '	0101 1110	Circumflex
95 #137	101 1111	_	SHIFT _	0101 1111	Underline
96 #140	110 0000	,	,	0110 0000	Grave accent
97 #141	110 0001	a	A	0110 0001	
98 #142	110 0010	b	B	0110 0010	
99 #143	110 0011	c	C	0110 0011	
100 #144	110 0100	d	D	0110 0100	
101 #145	110 0101	e	E	0110 0101	
102 #146	110 0110	f	F	0110 0110	
103 #147	110 0111	g	G	0110 0111	
104 #150	110 1000	h	H	0110 1000	
105 #151	110 1001	i	I	0110 1001	
106 #152	110 1010	j	J	0110 1010	
107 #153	110 1011	k	K	0110 1011	
108 #154	110 1100	l	L	0110 1100	
109 #155	110 1101	m	M	0110 1101	
110 #156	110 1110	n	N	0110 1110	
111 #157	110 1111	o	O	0110 1111	
112 #160	111 0000	p	P	0111 0000	
113 #161	111 0001	q	Q	0111 0001	
114 #162	111 0010	r	R	0111 0010	
115 #163	111 0011	s	S	0111 0011	
116 #164	111 0100	t	T	0111 0100	
117 #165	111 0101	u	U	0111 0101	
118 #166	111 0110	v	V	0111 0110	
119 #167	111 0111	w	W	0111 0111	
120 #170	111 1000	x	X	0111 1000	
121 #171	111 1001	y	Y	0111 1001	
122 #172	111 1010	z	Z	0111 1010	
123 #173	111 1011	{	{	0111 1011	Left brace
124 #174	111 1100		SHIFT	0111 1100	Vertical separator
125 #175	111 1101	}	SHIFT }	0111 1101	Right brace
126 #176	111 1110	-	SHIFT -	0111 1110	Tilde
127 #177	111 1111	DEL	DEL/REJ	0111 1111	Delete code

*Non ASCII codes*

CTRL SETUP	0000 0110
SETUP	0000 1011
NO SCRL	0000 1100
CTRL NO SCRL	0000 1110
ACC/ESC	0001 1011
CTRL!	0001 1100
CTRL!	0001 1101

Cursor key  
Cursor key

<i>Keystroke(s)</i>	<i>Raw key value</i>	<i>Comments</i>
CTRL-	0001 1110	Cursor key
CTRL→	001 1111	Cursor key
BREAK	1000 0000	
Cursor!	1000 0000	
Cursor!	1000 0000	
SHIFT BREAK	1000 0001	
Cursor←	1000 0001	
CTRL BREAK	1000 0010	
Cursor←	000 0010	
CTRL SHIFT BREAK	1000 0011	
PF1	1000 0100	Function key
PF2	1000 0101	Function key
PF3	1000 0110	Function key
CTRL HELP	1000 0111	
CTRL BACKSPACE	1000 1000	
CTRL TAB	1000 1001	
PF4	1000 1011	Function key
ENTER	1000 1100	
CTRL RETURN	1000 1101	
,	1001 0010	
—	1001 0011	
CTRL OOPS	1001 0101	
0	1001 0110	
1	1001 0111	
2	1001 1000	
3	1001 1001	
4	1001 1010	
CTRL ACC/ESC	1001 1011	
5	1001 1100	
6	1001 1101	
7	1001 1110	
8	1001 1111	
9	1010 0000	
CTRL SHIFT !	1010 0001	
CTRL SHIFT "	1010 0010	
CTRL SHIFT #	1010 0011	
CTRL SHIFT \$	010 0100	
CTRL SHIFT %	1010 0101	
CTRL SHIFT &	1010 0110	
CTRL '	1010 0111	
CTRL -	1010 1000	
CTRL SHIFT )	1010 1001	
CTRL SHIFT *	1010 1010	
CTRL SHIFT (	1010 1011	
CTRL SHIFT +	1010 1011	
CTRL ,	1010 1100	
CTRL -	1010 1101	
CTRL .	1010 1110	
CTRL /	1010 1111	
CTRL 0	1011 0000	
CTRL 1	1011 0001	
CTRL 2	1011 0010	
CTRL 3	1011 0011	
CTRL 4	1011 0100	
CTRL 5	1011 0101	
CTRL 6	1011 0110	
CTRL 7	1011 0111	
CTRL 8	1011 1000	
CTRL 9	1011 1001	
CTRL LF	1011 1010	
CTRL SHIFT :	1011 1010	
CTRL :	1011 1011	
CTRL SHIFT <	1011 1100	
CTRL =	1011 1101	
CTRL SHIFT >	1011 1110	
CTRL SHIFT ?	1011 1111	
CTRL SHIFT A	1100 0001	
CTRL SHIFT B	1100 0010	
CTRL SHIFT C	1100 0011	
CTRL SHIFT D	1100 0100	
CTRL SHIFT E	1100 0101	
CTRL SHIFT F	1100 0110	
CTRL SHIFT G	1100 0111	
CTRL SHIFT H	1100 1000	
CTRL SHIFT I	1100 1001	
CTRL SHIFT J	1100 1010	
CTRL SHIFT K	1100 1011	

<i>Keystroke(s)</i>	<i>Raw key value</i>	<i>Comments</i>
CTRL SHIFT L	1100 1100	
CTRL SHIFT M	1100 1101	
CTRL SHIFT N	1100 1110	
CTRL SHIFT O	1100 1111	
CTRL SHIFT P	1101 0000	
CTRL SHIFT Q	1101 0001	
CTRL SHIFT R	1101 0010	
CTRL SHIFT S	1101 0011	
CTRL SHIFT T	1101 0100	
CTRL SHIFT U	1101 0101	
CTRL SHIFT V	1101 0110	
CTRL SHIFT W	1101 0111	
CTRL SHIFT X	1101 1000	
CTRL SHIFT Y	1101 1001	
CTRL SHIFT Z	1101 1010	
CTRL SHIFT {	1111 1011	
CTRL SHIFT	1111 1100	
CTRL SHIFT }	1111 1101	
CTRL SHIFT ~	1111 1110	
CTRL DEL/REJ	1111 1111	



The following table lists the diagnostic display codes due to an incorrect boot.

<i>Display</i>	<i>Next event/failure</i>	<i>Possible faults</i>
150	Bootload not loaded correctly	Corrupt bootstrap
151	Bootload did not complete	Corrupt bootstrap or incorrect bootstrap (PERQ2 boot used on PERQ1 machine or vice versa)
152	Keyboard configure timed out	
153	Keyboard response not ACK or DATA	
154	Message queue invalid	
155	No interpreter for that key	Wrong button pressed during booting
156	No system for that key	Wrong button pressed during booting
157	Fixed disc error	Fixed disc
158	Floppy read error	Floppy disc
159	Microcode file length wrong	Microcode file
160	Checksum error in microcode	Microcode file
161	Checksum error in C-code	PNX file
162 to 168	Bad channel interrupts	Processor
169	Extended microcode on 4K machine	
170	Fixed disc response not ACK	Fixed disc (PERQ 1 only)
171	Fixed disc seek timeout	Fixed disc
172	Fixed disc seek error	Fixed disc
173	Fixed disc failed to power up	Fixed disc
174	Fixed disc wouldn't restore to COHO	Fixed disc (PERQ 2 only)
180	Floppy seek error	Floppy disc
181	Floppy seek timed out	Floppy disc
182	Unexpected floppy response	Floppy disc
194	Floppy Blkdata count odd	Floppy disc
196	Floppy read error	Floppy disc. could mean floppy disc continues bootstrap but no filestore

<i>Display</i>	<i>Next event/failure</i>	<i>Possible faults</i>
200	Z80 -> PERQ interrupt from unknown device ( $\geq 16$ or 0)	
200+N	(N<16) Interrupt from unknown or unexpected device (N)	
255	Loading successfully completed	None

The following two articles, the first on SCCS, the Source Code Control System, the second on *fsck(1)*, the file system consistency checker, are extracted from Volume 2B of the *UPM* for UNIX System 3. They are reproduced in their entirety exactly as printed in the *UPM*.



**SCCS User's Guide****SOURCE CODE CONTROL SYSTEM****USERS GUIDE**

**L. E. Bonanni  
C. A. Salemi**

**Bell Telephone Laboratories, Incorporated**



**SCCS User's Guide****Source Code Control System  
User's Guide**

1.	INTRODUCTION .....
2.	SCCS FOR BEGINNERS .....
2.1	Terminology
2.2	Creating an SCCS File - The "admin" Command
2.3	Retrieving a File - The "get" Command
2.4	Recording Changes - The "delta" Command
2.5	More about the "get" Command
2.6	The "help" Command
3.	HOW DELTAS ARE NUMBERED .....
4.	SCCS COMMAND CONVENTIONS .....
5.	SCCS COMMANDS .....
5.1	get
5.2	delta
5.3	admin
5.4	prs
5.5	help
5.6	rmdel
5.7	cdc
5.8	what
5.9	sccsdiff
5.10	comb
5.11	val
6.	SCCS FILES .....
6.1	Protection
6.2	Format
6.3	Auditing
	REFERENCES .....



**SCCS User's Guide****LIST OF FIGURES**

- Figure 1. Evolution of an SCCS File .....
- Figure 2. Tree Structure with Branch Deltas .....
- Figure 3. Extending the Branching Concept .....

## SCCS User's Guide

### Source Code Control System User's Guide

*L. E. Bonanni*

Bell Laboratories

Piscataway, New Jersey 08854

*C. A. Salemi*

Bell Laboratories

Piscataway, New Jersey 08854

## 1. INTRODUCTION

The Source Code Control System (SCCS) is a collection of commands that help individuals or projects control and account for changes to files of text (typically, the source code and documentation of software systems). It is convenient to conceive of SCCS as a custodian of files; it allows retrieval of particular versions of the files, administers changes to them, controls updating privileges to them, and records who made each change, when and where it was made, and why. This is important in environments in which programs and documentation undergo frequent changes (because of maintenance and/or enhancement work), inasmuch as it is sometimes desirable to regenerate the version of a program or a document as it was before changes were applied to it. Obviously, this could be done by keeping copies (on paper or other media), but this quickly becomes unmanageable and wasteful as the number of programs and documents increases. SCCS provides an attractive solution because it stores on disc the original file and, whenever changes are made to it, stores only the *changes*; each set of changes is called a "delta".

This document, together with relevant portions of *UPM(1)*, is a complete user's guide to SCCS. This manual contains the following sections:

- *SCCS for Beginners*: How to make an SCCS file, how to update it, and how to retrieve a version thereof.
- *How Deltas Are Numbered*: How versions of SCCS files are numbered and named.
- *SCCS Command Conventions*: Conventions and rules generally applicable to all SCCS commands
- *SCCS Commands*: Explanation of all SCCS commands, with discussions of the more useful arguments
- *SCCS Files*: Protection, format, and auditing of SCCS files, including a discussion of the differences between using SCCS as an individual and using it as a member of a group or project. The role of a "project SCCS administrator" is introduced.

## 2. SCCS FOR BEGINNERS

It is assumed that the reader knows how to log onto a system, create files, and use the text editor, *ed(1)*. A number of terminal-sessions fragments are presented below. All of them should be tried; the best way to learn SCCS is to use it.

To supplement the material in this manual, the detailed SCCS command descriptions (appearing in *UPM(1)*) should be consulted. Section 5 below contains a list of all the SCCS commands. For the time being, however, only basic concepts will be discussed.

### 2.1 Terminology

Each SCCS file is composed of one or more sets of changes applied to the null (empty) version of the file, with each set of changes usually depending on all previous sets. Each set of changes is called a "delta" and is assigned a name, called the SCCS IDentification string (SID), composed of at most four components, only the first two of which will concern us for now; these are the "release" and "level" numbers, separated by a period. Hence, the first delta is called "1.1", the second "1.2", the third "1.3", etc. The release number can also be changed allowing, for example, deltas "2.1", "3.19", etc. The change in the release number usually indicates a major change to the file.

## SCCS User's Guide

Each delta of an SCCS file defines a particular version of the file. For example, delta 1.5 defines version 1.5 of the SCCS file, obtained by applying to the null (empty) version of the file the changes that constitute deltas 1.1, 1.2, etc., up to and including delta 1.5 itself, in that order.

### 2.2 Creating an SCCS File - The "admin" Command

Consider, for example, a file called "lang" that contains a list of programming languages:

```
c  
pl/i  
fortran  
cobol  
algol
```

We wish to give custody of this file to SCCS. The following *admin* command (which is used to administer SCCS files) creates an SCCS file and initializes delta 1.1 from the file "lang":

```
admin - ilang s.lang
```

All SCCS files *must* have names that begin with "s.", hence, "s.lang". The *-i* keyletter, together with its value "lang", indicates that *admin* is to create a new SCCS file and *initialize* it with the contents of the file "lang". This initial version is a set of changes applied to the null SCCS file; it is delta 1.1.

The *admin* command replies:

```
No id keywords (cm7)
```

This is a warning message (which may also be issued by other SCCS commands) that is to be ignored for the purpose of this section. Its significance is described in Section 5.1 below. In the following examples, this warning message is not shown, although it may actually be issued by the various commands.

The file "lang" should be removed (because it can be easily reconstructed by using the *get* command, below):

```
rm lang
```

### 2.3 Retrieving a File - The "get" Command

The command:

```
get s.lang
```

causes the creation (retrieval) of the latest version of file "s.lang" and prints the following messages:

```
1.1  
5 lines
```

This means that *get* retrieved version 1.1 of the file, which is made up of 5 lines of text. The retrieved text is placed in a file whose name is formed by deleting the "s." prefix from the name of the SCCS file; hence, the file "lang" is created.

The above *get* command simply creates the file "lang" read-only, and keeps no information whatsoever regarding its creation. On the other hand, in order to be able to subsequently apply changes to an SCCS file with the *delta* command (see below), the *get* command must be informed of your intention to do so. This is done as follows:

```
get -e s.lang
```

The *-e* keyletter causes *get* to create a file "lang" for both reading and writing (so that it may be edited) and places certain information about the SCCS file in another new file, called the *p-file*, that will be read by the *delta* command. The *get* command prints the same messages as before, except that the SID of the version to be created through the use of *delta* is also issued.

## SCCS User's Guide

For example:

```
get -e s.lang
1.1
new delta 1.2
5 lines
```

The file "lang" may now be changed, for example, by:

```
ed lang
27
$ a
snobol
ratfor
.
w
41
q
```

### 2.4 Recording Changes-The "delta" Command

In order to record within the SCCS file the changes that have been applied to "lang", execute:

```
delta s.lang
```

*Delta* prompts with:

```
comments?
```

the response to which should be a description of why the changes were made; for example:

```
comments? added more languages
```

*Delta* then reads the *p-file*, and determines what changes were made to the file "lang". It does this by doing its own *get* to retrieve the original version, and by applying *diff(1)*<sup>1</sup> to the original version and the edited version.

When this process is complete, at which point the changes to "lang" have been stored in "s.lang", *delta* outputs:

```
1.2
2 inserted
0 deleted
5 unchanged
```

The number "1.2" is the name of the delta just created, and the next three lines of output refer to the number of lines in the file "s.lang".

### 2.5 More about the "get" Command

As we have seen:

```
get s.lang
```

retrieves the latest version (now 1.2) of the file "s.lang". This is done by starting with the original version of the file and successively applying deltas (the changes) in order, until all have been applied.

For our example, the following commands are all equivalent:

```
get s.lang
get -r1 s.lang
get -r1.2 s.lang
```

---

<sup>1</sup> All references of the form *name (n)* refer to item *name* in command writeup section *N* of *UPM(1)*

## SCCS User's Guide

The numbers following the `-r` keyletter are SIDS (see Section 2.1 above). Note that omitting the level number of the SID (as in the second example above) is equivalent to specifying the *highest* level number that exists within the specified release. Thus, the second command requests the retrieval of the latest version in release 1, namely 1.2. The third command specifically requests the retrieval of a particular version, in this case, also 1.2.

Whenever a truly major change is made to a file, the significance of that change is usually indicated by changing the *release* number (first component of the SID) of the delta being made. Since normal, automatic, numbering of deltas proceeds by incrementing the level number (second component of the SID), we must indicate to SCCS that we wish to change the release number. This is done with the `get` command:

```
get -e -r2 s.lang
```

Because release 2 does not exist, `get` retrieves the latest version *before* release 2; it also interprets this as a request to change the release number of the delta we wish to create to 2, thereby causing it to be named 2.1, rather than 1.3. This information is conveyed to *delta* via the *p-file*. `Get` then outputs:

```
1.2
new delta 2.1
7 lines
```

which indicates that version 1.2 has been retrieved and that 2.1 is the version *delta* will create. If the file is now edited, for example, by:

```
ed lang
41
/cobol/d
w
35
q
```

and *delta* executed:

```
delta s.lang
comments? deleted cobol from list of languages
```

we will see, by *delta*'s output, that version 2.1 is indeed created:

```
2.1
0 inserted
1 deleted
6 unchanged
```

Deltas may now be created in release 2 (deltas 2.2, 2.3, etc.), or another new release may be created in a similar manner. This process may be continued as desired.

### 2.6 The "help" Command

If the command:

```
get abc
```

is executed, the following message will be output:

```
ERROR [abc]: not an SCCS file (col)
```

The string "col" is a code for the diagnostic message, and may be used to obtain a fuller explanation of that message by use of the `help` command:

```
help col
```

This produces the following output:

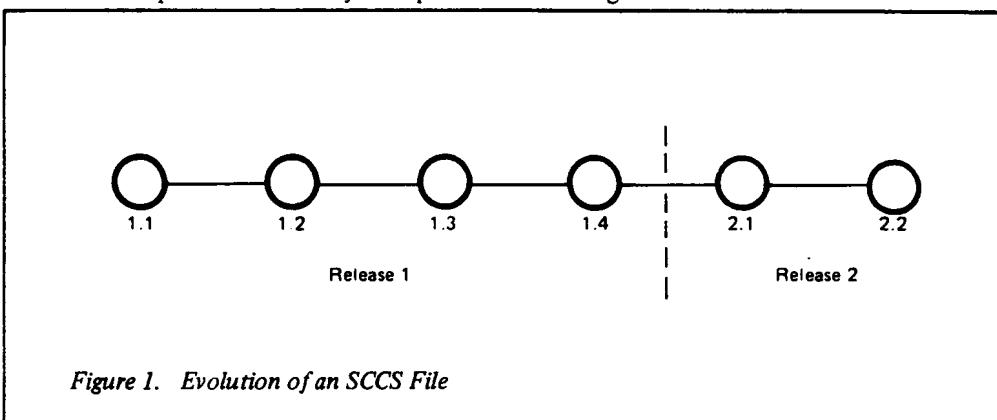
```
col:
"not an SCCS file"
A file that you think is an SCCS file
does not begin with the characters "s"
```

## SCCS User's Guide

Thus, *help* is a useful command to use whenever there is any doubt about the meaning of an SCCS message. Fuller explanations of almost all SCCS messages may be found in this manner.

### 3. HOW DELTAS ARE NUMBERED

It is convenient to conceive of the deltas applied to an SCCS file as the nodes of a tree, in which the root is the initial version of the file. The root delta (node) is normally named "1.1" and successor deltas (nodes) are named "1.2", "1.3", etc. The components of the names of the deltas are called the "release" and the "level" numbers, respectively. Thus, normal naming of successor deltas proceeds by incrementing the level number, which is performed automatically by SCCS whenever a delta is made. In addition, the user may wish to change the release number when making a delta, to indicate that a major change is being made. When this is done, the release number also applies to all successor deltas, unless specifically changed again. Thus, the evolution of a particular file may be represented as in Figure 1.



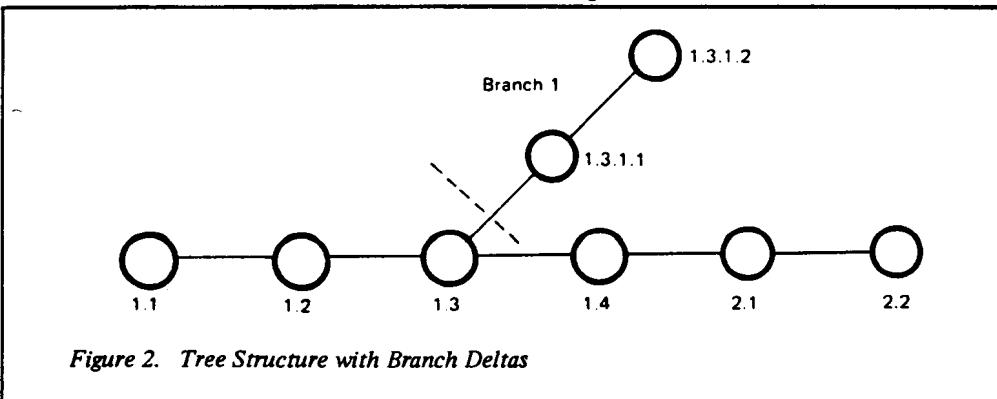
Such a structure may be termed the "trunk" of the SCCS tree. It represents the normal *sequential* development of an SCCS file, in which changes that are part of any given delta are dependent upon *all* the preceding deltas.

However, there are situations in which it is necessary to cause a *branching* in the tree, in that changes applied as part of a given delta are *not* dependent upon all previous deltas. As an example, consider a program which is in production use at version 1.3, and for which development work on release 2 is already in progress. Thus, release 2 may already have some deltas, precisely as shown in Figure 1. Assume that a production user reports a problem in version 1.3, and that the nature of the problem is such that it cannot wait to be repaired in release 2. The changes necessary to repair the trouble will be applied as a delta to version 1.3 (the version in production use). This creates a new version that will then be released to the user, but will *not* affect the changes being applied for release 2 (ie., deltas 1.4, 2.1, 2.2, etc.).

The new delta is a node on a "branch" of the tree, and its name consists of *four* components, namely, the release and the level numbers, as with trunk deltas, plus the "branch" and "sequence" numbers, as follows:

release.level.branch.sequence

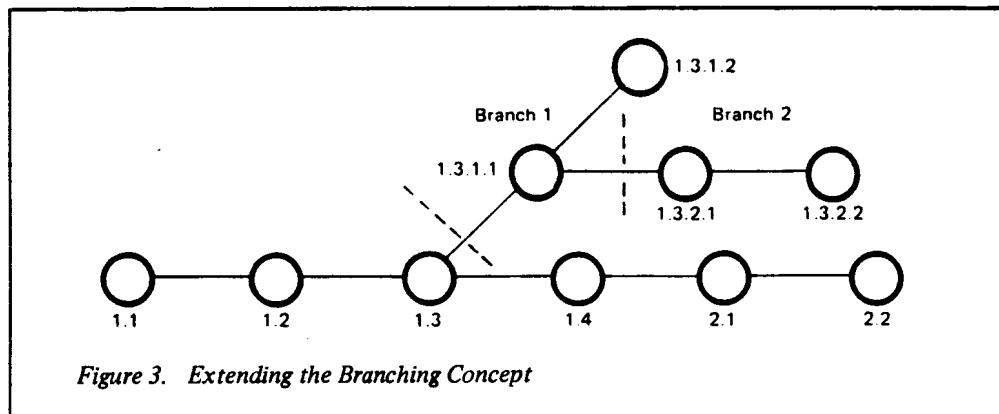
The *branch* number is assigned to each branch that is a descendant of a particular trunk delta, with the first such branch being 1, the next one 2, and so on. The *sequence* number is assigned, in order, to each delta on a *particular branch*. Thus, 1.3.1.2 identifies the second delta of the first branch that derives from delta 1.3. This is shown in Figure 2.



## SCCS User's Guide

The concept of branching may be extended to any delta in the tree; the naming of the resulting deltas proceeds in the manner just illustrated.

Two observations are of importance with regard to naming deltas. First, the names of trunk deltas contain exactly two components, and the names of branch deltas contain exactly four components. Second, the first two components of the name of branch deltas are always those of the ancestral trunk delta, and the branch component is assigned in the order of creation of the branch, independently of its location relative to the trunk delta. Thus, a branch delta may always be identified as such from its name. Although the ancestral trunk delta may be identified from the branch delta's name, it is *not* possible to determine the *entire* path leading from the trunk delta to the branch delta. For example, if delta 1.3 has one branch emanating from it, all deltas on that branch will be named 1.3.1.n. If a delta on this branch then has another branch emanating from it, all deltas on the new branch will be named 1.3.2.n (see Figure 3). The only information that may be derived from the name of delta 1.3.2.2 is that it is the *chronologically* second delta on the *chronologically* second branch whose *trunk* ancestor is delta 1.3. In particular, it is *not* possible to determine from the name of delta 1.3.2.2 all of the deltas between it and its trunk ancestor (1.3).



*Figure 3. Extending the Branching Concept*

It is obvious that the concept of branch deltas allows the generation of arbitrarily complex tree structures. Although this capability has been provided for certain specialized uses, it is strongly recommended that the SCCS tree be kept as simple as possible, because comprehension of its structure becomes extremely difficult as the tree becomes more complex.

### 4. SCCS COMMAND CONVENTIONS

This section discusses the conventions and rules that apply to SCCS commands. These rules and conventions are generally applicable to *all* SCCS commands, except as indicated below. SCCS commands accept two types of arguments: *keyletter* arguments and *file* arguments.

*Keyletter* arguments (hereafter called simply "keyletters") begin with a minus sign (-), followed by a lower-case alphabetical character, and, in some cases, followed by a value. These keyletters control the execution of the command to which they are supplied.

*File* arguments (which may be names of files and/or directories) specify the file(s) that the given SCCS command is to process; naming a directory is equivalent to naming *all* the SCCS files within the directory. Non-SCCS files and unreadable<sup>2</sup> files in the named directories are silently ignored.

In general, file arguments may *not* begin with a minus sign. However, if the name "--" (a lone minus sign) is specified as an argument to a command, the command reads the standard input for lines and takes each line as the *name* of an SCCS file to be processed. The standard input is read until end-of-file. This feature is often used in pipelines (see *UPM(1)*) with, for example, the *find(1)* or *ls(1)* commands. Again, names of non-SCCS files and of unreadable files are silently ignored.

<sup>2</sup> Because of permission modes (see *chmod(1)*).

## SCCS User's Guide

All keyletters specified for a given command apply to *all* file arguments of that command. All keyletters are processed before any file arguments, with the result that the placement of keyletters is arbitrary (i.e., keyletters may be interspersed with file arguments). File arguments, however are processed left to write.

Somewhat different argument conventions apply to the *help*, *what*, *sccsdiff*, and *val* commands (see sections 5.5, 5.8, 5.9, and 5.11).

Certain actions of various SCCS commands are controlled by *flags* appearing in SCCS files. Some of these *flags* are discussed below. For a complete description of all such flags, see *admin(1)*.

The distinction between the *real user* (see *passwd(1)*) and the effective user of the system is of concern in discussing various actions of SCCS commands. For the present, it is assumed that both the real user and the effective user are one and the same (i.e., the user who is logged into a system); this subject is further discussed in Section 6.1.

All SCCS commands that modify an SCCS file do so by writing a temporary copy, called the *x-file*, which ensures that the SCCS file will not be damaged should processing terminate abnormally. The name of the *x-file* is formed by replacing the "s." of the SCCS file name with "x.". When processing is complete, the old SCCS file is removed and the *x-file* is renamed to be the SCCS file. The *x-file* is created in the directory containing the SCCS file, is given the same mode (see *chmod(1)*) as the SCCS file, and is owned by the effective user.

To prevent simultaneous updates to an SCCS file, commands that modify SCCS files create a *lock-file*, called the *z-file*, whose name is formed by replacing the "s." of the SCCS file name with "z.". The *z-file* contains the *process number* (see *UPM(1)*) of the command that creates it, and its existence is an indication to other commands that that SCCS file is being updated. Thus, other commands that modify SCCS files will not process an SCCS file if the corresponding *z-file* exists. The *z-file* is created with mode 444 (read only) in the directory containing the SCCS file, and is owned by the effective user. This file exists only for the duration of the execution of the command that creates it. In general, users can ignore *x-files* and *z-files*; they may be useful in the event of system crashes or similar situations.

SCCS commands produce diagnostics (on the standard error output (see *UPM(1)*) of the form:

ERROR [name-of-file-being-processed]: message text (code)

The *code* in parentheses may be used as an argument to the *help* command (see Section 5.5) to obtain a further explanation of the diagnostic message.

Detection of a fatal error during the processing of a file causes the SCCS command to terminate processing of *that* file and to proceed with the next file, in order, if more than one file has been named.

## 5. SCCS COMMANDS

This section describes the major features of all the SCCS commands. Detailed descriptions of the commands and of all their arguments are given in the *UPM(1)*, and should be consulted for further information. The discussion below covers only the more common arguments of the various SCCS commands.

Because the commands *get* and *delta* are the most frequently used, they are presented first. The other commands follow in approximate order of importance.

The following is a summary of all the SCCS commands and of their major functions:

get	Retrieves versions of SCCS files.
delta	Applies changes (deltas) to the text of SCCS files, ie., creates new versions.
admin	Creates SCCS files and applies changes to parameters of SCCS files.
prs	Prints portions of an SCCS file in user specified format.
help	Gives explanations of diagnostic messages.

## SCCS User's Guide

<b>rmDEL</b>	Removes a delta from an SCCS file; allows the removal of deltas that were created by mistake.
<b>cdc</b>	Changes the commentary associated with a delta.
<b>what</b>	Searches file(s) for all occurrences of a special pattern and prints out what follows it; is useful in finding identifying information inserted by the <i>get</i> command.
<b>sccsdiff</b>	Shows the differences between any two versions of an SCCS file.
<b>comb</b>	Combines two or more consecutive deltas of an SCCS file into a single delta; often reduces the size of the SCCS file.
<b>val</b>	Validates an SCCS file.

### 5.1 **get**

The *get* command creates a text file that contains a particular version of an SCCS file. The particular version is retrieved by beginning with the initial version, and then applying deltas, in order, until the desired version is obtained. The created file is called the *g-file*: its name is formed by removing the "s." from the SCCS file name. The *g-file* is created in the current directory (see *UPM(1)*) and is owned by the real user. The mode assigned to the *g-file* depends on how the *get* command is invoked, as discussed below.

The most common invocation of *get* is:

```
get s.abc
```

which normally retrieves the latest version on the trunk of the SCCS file tree, and produces (for example) on the standard output:

```
1.3
67 lines
No id keywords (cm7)
```

which indicates that:

- 1 Version 1.3 of file "s.abc" was retrieved (1.3 is the latest trunk delta).
- 2 This version has 67 lines of text.
- 3 No ID keywords were substituted in the file (see Section 5.1.1 for a discussion of ID keywords).

The generated *g-file* (file "abc") is given mode 444 (read-only), since this particular way of invoking *get* is intended to produce *g-files* only for inspection, compilation, etc., and *not* for editing (ie., *not* for making deltas).

In the case of several file arguments (or directory-name arguments), similar information is given for each file processed, but the SCCS file name precedes it. For example:

```
get s.abc s.def
```

produces:

```
s.abc:
1.3
67 lines
No valid keywords (cm7)
```

```
s.def:
1.7
85 lines
No id keywords (cm7)
```

## SCCS User's Guide

### 5.1.1 ID keywords

In generating a **g-file** to be used for compilation, it is useful and informative to record the date and time of creation, the version retrieved, the module's name, etc., within the *g-file*, so as to have this information appear in a load module when one is eventually created. SCCS provides a convenient mechanism for doing this automatically. *Identification (ID) keywords* appearing anywhere in the generated file are replaced by appropriate values according to the definitions of these ID keywords. The format of an ID keyword is an upper-case letter enclosed by percent signs (%). For example:

%I%

is defined as the ID keyword that is replaced by the SID of the retrieved version of a file. Similarly, %H% is defined as the ID keyword for the current date (in the form "mm/dd/yy"), and %M% is defined as the name of the *g-file*. Thus, executing *get* on an SCCS file that contains the PL/I declaration:

DCL ID CHAR(100) VAR INIT('%M% %I% %H%');

gives (for example) the following:

DCL ID CHAR(100) VAR INIT('MODNAME 2.3 07/07/77');

When no ID keywords are substituted by *get*, the following message is issued:

No id keywords (cm7)

This message is normally treated as a warning by *get*, although the presence of the i flag in the SCCS file causes it to be treated as an error (see Section 5.2 for further information).

For a complete list of the approximately twenty ID keywords provided, see *get*(1).

### 5.1.2 Retrieval of Different Versions

Various keyletters are provided to allow the retrieval of other than the default version of an SCCS file. Normally, the default version is the most recent delta of the highest-numbered release on the *trunk* of the SCCS file tree. However, if the SCCS file being processed has a d (default SID) flag, the SID specified as the value of this flag is used as a default. The default SID is interpreted in exactly the same way as the value supplied with the -r keyletter of *get*.

The -r keyletter is used to specify an SID to be retrieved, in which case the d (default SID) flag (if any) is ignored. For example:

get -r1.3 s.abc

retrieves version 1.3 of file "s.abc", and produces (for example) on the standard output:

1.3  
64 lines

A branch delta may be retrieved similarly:

get -r1.5.2.3 a.abc

which produces (for example) on the standard output:

1.5.2.3  
234 lines

When a two- or four-component SID is specified as a value for the -r keyletter (as above) and the particular version does not exist in the SCCS file, an error message results. Omission of the level number, as in:

get -r3 s.abc

causes retrieval of the *trunk* delta with the highest level number within the given release, if the given release exists. Thus, the above command might output:

## SCCS User's Guide

3.7

213 lines

If the given release does not exist, *get* retrieves the *trunk* delta with the highest level number within the highest-numbered existing release that is lower than the given release. For example, assuming release 9 does not exist in file "s.abc", and the release 7 is actually the highest-numbered release below 9, execution of:

`get -r9 s.abc`

might produce:

7.6

420 lines

which indicates that trunk delta 7.6 is the latest version of file "s.abc" below release 9. Similarly, omission of the sequence number, as in:

`get -r4.3.2 s.abc`

results in the retrieval of the branch delta with the highest sequence number on the given branch, if it exists. (If the given branch does not exist, an error message results.) This might result in the following output:

4.3.2.8

89 lines

The *-t* keyletter is used to retrieve the latest ("top") version in a particular *release* (i.e., when no *-r* keyletter is supplied, or when its value is simply a *release* number). The latest version is defined as that delta which was produced most recently, independent of its location on the SCCS file tree. Thus, if the most recent delta in *release* 3 is 3.5.

`get -r3 -t s.abc`

might produce:

3.5

59 lines

However, if branch delta 3.2.1.5 were the latest (created after delta 3.5), the same command might produce:

3.2.1.5

46 lines

### 5.1.3 Retrieval with Intent to Make a Delta

Specification of the *-e* keyletter to the *get* command is an indication of the intent to make a delta, and as such, its use is restricted. The presence of this keyletter causes *get* to check:

- 1 The user *list* (which is the list of *login* names and/or *group* IDs of users allowed to make deltas (see Section 6.2) to determine if the *login* name or *group* ID of the user executing *get* is on the list. Note that a *null* (empty) user list behaves as if it contained *all* possible *login* names.
- 2 That the *release* (*R*) of the version being retrieved satisfies the relation:

$$\text{floor} \leq R \leq \text{ceiling}$$

to determine if the release being accessed is a protected release. The *floor* and *ceiling* are specified as *flags* in the SCCS file.

- 3 That the *release* (*R*) is not *locked* against editing. The *lock* is specified as a *flag* in the SCCS file.
- 4 Whenever or not *multiple concurrent edits* are allowed for the SCCS file as specified by the *j* flag in the SCCS file (multiple concurrent edits are described in Section 5.1.5).

A failure of any of the first three conditions causes the processing of the corresponding SCCS file to terminate.

## SCCS User's Guide

If the above checks succeed, the `-e` keyletter causes the creation of a *g-file* in the current directory with mode 644 (readable by everyone, writable only by the owner) owned by the real user. If a *writable g-file* already exists, *get* terminates with an error. This is to prevent inadvertent destruction of a *g-file* that already exists and is being edited for the purpose of making a delta.

Any ID keywords appearing in the *g-file* are *not* substituted by *get* when the `-e` keyletter is specified, because the generated *g-file* is to be subsequently used to create another delta, and replacement of ID keywords would cause them to be permanently changed within the SCCS file. In view of this, *get* does not need to check for the presence of ID keywords within the *g-file*, so that the message:

No id keywords (cm7)

is never output when *get* is invoked with the `-e` keyletter.

In addition, the `-e` keyletter causes the creation (or updating) of a *p-file*, which is used to pass information to the *delta* command (see Section 5.1.4).

The following is an example of the use of the `-e` keyletter:

```
get -e s.abc
```

which produces (for example) on the standard output:

```
1.3  
new delta 1.4  
67 lines
```

If the `-r` and/or `-t` keyletters are used together with the `-e` keyletter, the version retrieved for editing is a specified by the `-r` and/or `-t` keyletters.

The keyletters `-i` and `-x` may be used to specify a list (see *get(1)* for the syntax of such a list) of deltas to be *included* and *excluded*, respectively, by *get*. Including a delta means forcing the changes that constitute the particular delta to be included in the retrieved version. This is useful if one wants to apply the same changes to more than one version of the SCCS file. Excluding a delta means forcing it to be *not* applied. This may be used to undo, in the version of the SCCS file to be created, the effects of a previous delta. Whenever deltas are included or excluded, *get* checks for possible interference between such deltas and those deltas that are normally used in retrieving the particular version of the SCCS file. (Two deltas can interfere, for example, when each one changes the same line of the retrieved *g-file*.) Any interference is indicated by a warning that shows the range of lines within the retrieved *g-file* in which the problem may exist. The user is expected to examine the *g-file* to determine whether a problem actually exists, and to take whatever corrective measures (if any) are deemed necessary (e.g., edit the file).

Note: *The -i and -x keyletters should be used with extreme care.*

The `-k` keyletter is provided to facilitate regeneration of a *g-file* that may have been accidentally removed or ruined subsequent to the execution of *get* with the `-e` keyletter, or to simply generate a *g-file* in which the replacement of ID keywords has been suppressed. Thus, a *g-file* generated by the `-k` keyletter is identical to one produced by *get* executed with the `-e` keyletter. However, no processing related to the *p-file* takes place.

### 5.1.4 Concurrent Edits of Different SIDs

The ability to retrieve different versions of an SCCS file allows a number of deltas to be "in progress" at any given time. This means that a number of *get* commands with the `-e` keyletter may be executed on the same file, provided that no two executions retrieve the same version (unless multiple concurrent edits are allowed, see section 5.1.5).

## SCCS User's Guide

The *p-file* (which is created by the *get* command invoked with the *-e* keyletter) is named by replacing the "s." in the SCCS file name with "p.". It is created in the directory containing the SCCS file, is given mode 644 (readable by everyone, writable only by the owner), and is owned by the effective user. The *p-file* contains the following information for each delta that is still "in progress":<sup>3</sup>

- . The SID of the retrieved version.
- . The SID that will be given to the new delta when it is created.
- . The login name of the real user executing *get*.

The first execution of "get -e" causes the *creation* of the *p-file* for the corresponding SCCS file. Subsequent executions only *update* the *p-file* by inserting a line containing the above information. Before inserting this line, however, *get* checks that no entry already in the *p-file* specifies as already retrieved the SID of the version to be retrieved, unless multiple concurrent edits are allowed.

If both checks succeed, the user is informed that other deltas are in progress, and processing continues. If either check fails, an error message results. It is important to note that the various executions of *get* should be carried out from different directories. Otherwise, only the first execution will succeed, since subsequent executions would attempt to over-write a *writable g-file*, which is an SCCS error condition. In practice, such multiple executions are performed by different users,<sup>4</sup> so that this problem does not arise, since each user normally has a different working or home directory.

Table 1 shows, for the most useful cases, what version of an SCCS file is retrieved by *get*, as well as the SID of the version to be eventually created by *delta*, as a function of the SID specified to *get*.

### 5.1.5 Concurrent Edits of the Same SID

Under normal conditions, *gets* for editing (-e keyletter is specified) based on the same SID are not permitted to occur concurrently. That is, *delta* must be executed before a subsequent *get* for editing is executed at the same SID as the previous *get*. However, multiple concurrent edits (defined to be two or more *successive* executions of *get* for editing based on the same retrieved SID) are allowed if the *j* flag is set in the SCCS file. Thus:

```
get -e s.abc
1.1
new delta 1.2
5 lines
```

may be immediately followed by:

```
get -e s.abc
1.1
new delta 1.1.1.1
5 lines
```

without an intervening execution of *delta*. In this case, a *delta* command corresponding to the first *get* produces delta 1.2 (assuming 1.1 is the latest (most recent) trunk delta), and the *delta* command corresponding to the second *get* produces delta 1.1.1.1.

### 5.1.6 Keyletters That Affect Output

Specification of the *-p* keyletter causes *get* to write the retrieved text to the standard output, rather than to a *g-file*. In addition, all output normally directed to the standard output (such as the SID of the version retrieved and the number of lines retrieved) is directed instead to the standard error output. This may be used, for example, to create *g-files* with arbitrary names:

```
get -p s.abc > arbitrary-filename
```

---

3 Other information may be present, but is not of concern here. See *get(1)* for further information

4 See section 6.1 for a discussion of how different users are permitted to use SCCS commands on the same files.

**SCCS User's Guide****TABLE 1. Determination of New SID**

<i>Case</i>	<i>SID Specified*</i>	<i>-b Keyletter Used†</i>	<i>Other Conditions</i>	<i>SID Retrieved</i>	<i>SID of Delta to be Created</i>
1.	none↓	no	R defaults to mR	mR.mL	mR.(mL + 1)
2.	none↓	yes	R defaults to mR	mR.mL	mR.mL.(mB + 1).1
3.	R	no	R > mR	mR.mL	R.1§
4.	R	no	R = mR	mR.mL	mR.(mL + 1)
5.	R	yes	R > mR	mR.mL	mR.mL.(mB + 1).1
6.	R	yes	R = mR	mR.mL	mR.mL.(mB + 1).1
7.	R	-	R < mR and R does <i>not</i> exist	hR.mL**	hR.mL.(mB + 1).1
8.	R	-	Trunk successor in release > R and R exists	R.mL	R.mL.(mB + 1).1
9.	R.L	no	No trunk successor	R.L	R.(L + 1)
10.	R.L	yes	No trunk successor	R.L	R.L.(mB + 1).1
11.	R.L	-	Trunk successor in release ≥ R	R.L	R.L.(mB + 1).1
12.	R.L.B	no	No branch successor	R.L.B.mS	R.L.B.(mS + 1)
13.	R.L.B	yes	No branch successor	R.L.B.mS	R.L.(mB + 1).1
14.	R.L.B.S	no	No branch successor	R.L.B.S	R.L.B.(S + 1)
15.	R.L.B.S	yes	No branch successor	R.L.B.S	R.L.(mB + 1).1
16.	R.L.B.S	-	Branch successor	R.L.B.S	R.L.(mB + 1).1

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

† The -b keyletter is effective only if the b flag (see *admin(1)*) is present in the file. In this table, an entry of "-" means "irrelevant".

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

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

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

## SCCS User's Guide

The -p keyletter is particularly useful when used with the “!” or “\$” arguments of the *send(1)* command. For example:

```
send MOD=s.abc REL=3 compile
```

if the file “compile” contains:

```
//plicom job job-card-information
//stepl exec plickc
//pli.sysin dd *
-s
!get -p -rREL MOD
/*
//
```

will *send* the highest level of release 3 of file “s.abc”. Note that the line “-s”, which causes *send(1)* to make ID keyword substitutions before detecting and interpreting control lines, is necessary if *send(1)* is to substitute “s.abc” for MOD and “3” for REL in the line “!get -p -rREL MOD”.

The -s keyletter suppresses all output that is *normally* directed to the standard output. Thus, the SID of the retrieved version, the number of lines retrieved, etc., are not output. This does not, however, affect messages to the standard error output. This keyletter is used to prevent non-diagnostic messages from appearing on the user's terminal, and is often used in conjunction with the -p keyletter to “pipe” the output of *get*, as in:

```
get -p -s s.abc | nroff
```

The -g keyletter is supplied to suppress the actual retrieval of the text of a version of the SCCS file. This may be useful in a number of ways. For example, to verify the existence of a particular SID in an SCCS file, one may execute:

```
get -g -r4.3 s.abc
```

This outputs the given SID if it exists in the SCCS file, or it generates an error message, if it does not. Another use of the -g keyletter is in regenerating a *p-file* that may have been accidentally destroyed:

```
get -e -g s.abc
```

The -l keyletter causes the creation of an *l-file*, which is named by replacing the “s.” of the SCCS file name with “l.”. This file is created in the current directory, with mode 444 (read-only), and is owned by the real user. It contains a table (whose format is described in *get(1)*) showing which deltas were used in constructing a particular version of the SCCS file. For example:

```
get -r2.3 -l s.abc
```

generates an *l-file* showing which deltas were applied to retrieve version 2.3 of the SCCS file. Specifying a *value* of “p” with the -l keyletter, as in:

```
get -lp -r2.3 s.abc
```

causes the generated output to be written to the standard output rather than to the *l-file*. Note that the -g keyletter may be used with the -l keyletter to suppress the actual retrieval of the text.

The -m keyletter is of use in identifying, line by line, the changes applied to an SCCS file. Specification of this keyletter causes each line of the generated *g-file* to be preceded by the SID of the delta that caused that line to be inserted. The SID is separated from the text of the line by a tab character.

The -a keyletter causes each line of the generated *g-file* to be preceded by the value of the %M% ID keyword (see Section 5.1.1) and a tab character. The -n keyletter is most often used in a pipeline with *grep(1)*. For example, to find all lines that match a given pattern in the latest version of each SCCS file in a directory, the following may be executed:

```
get -p -a -s directory | grep pattern
```

## SCCS User's Guide

If both the -m and -n keyletters are specified, each line of the generated *g-file* is preceded by the value of the %M% ID keyword and a tab (this is the effect of the -n keyletter), followed by the line in the format produced by the -m keyletter. Because use of the -m keyletter and/or the -n keyletter causes the contents of the *g-file* to be modified, such a *g-file* must not be used for creating a delta. Therefore, neither the -m keyletter nor the -n keyletter may be specified together with the -e keyletter.

See *get(1)* for a full description of additional *get* keyletters.

### 5.2 delta

The *delta* command is used to incorporate the changes made to a *g-file* into the corresponding SCCS file, i.e., to create a delta, and, therefore, a new version of the file.

Invocation of the *delta* command requires the existence of a *p-file* (see Sections 5.1.3 and 5.1.4). *Delta* examines the *p-file* to verify the presence of an entry containing the user's login name. If none is found, an error message results. *Delta* also performs the same permission checks that *get* performs when invoked with the -e keyletter. If all checks are successful, *delta* determines what has been changed in the *g-file*, by comparing it (via *diff(1)*) with its own, temporary copy of the *g-file* as it was before editing. This temporary copy of the *g-file* is called the *d-file* (its name is formed by replacing the "s." of the SCCS file name with "d.") and is obtained by performing an internal *get* at the SID specified in the *p-file* entry.

The required *p-file* entry is the one containing the login name of the user executing *delta*, because the user who retrieved the *g-file* must be the one who will create the delta. However, if the login name of the user appears in more than one entry (i.e., the same user executed *get* with the -e keyletter more than once on the same SCCS file), the -r keyletter must be used with *delta* to specify an SID that uniquely identifies the *p-file* entry<sup>5</sup>. This entry is the one used to obtain the SID of the delta to be created.

In practice, the most common invocation of *delta* is:

```
delta s.abc
```

which prompts on the standard output (but only if it is a terminal):

```
comments?
```

to which the user replies with a description of why the delta is being made, terminating the reply with a newline character. The user's response may be up to 512 characters long, with newlines *not* intended to terminate the response escaped by "\".

If the SCCS file has a v flag, *delta* first prompts with:

```
MRs?
```

on the standard output. (Again, this prompt is printed only if the standard output is a terminal.) The standard input is then read for MR<sup>6</sup> numbers, separated by blanks and/or tabs, terminated in the same manner as the response to the prompt "comments?"

The -y and/or -m keyletters are used to supply the commentary (comments and MR numbers, respectively) on the command line, rather than through the standard input. For example:

```
delta -y"descriptive comment" -m"mrnum1 mrnum2" s.abc
```

- 
- 5 The specified may be either the SID retrieved by *get*, or the SID *delta* is to create.
  - 6 In a tightly controlled environment, it is expected that deltas are created only as a result of some trouble report, change request, trouble ticket, etc. (collectively called here Modification Requests, or MRs) and that it is desirable or necessary to record such MR number(s) within each delta. .

## SCCS User's Guide

In this case, the corresponding prompts are not printed, and the standard input is not read. The **-m** keyletter is allowed only if the SCCS file has a **v** flag. These keyletters are useful when *delta* is executed from within a *shell procedure* (see *sh(1)*).

The commentary (comments and/or MR numbers), whether solicited by *delta* or supplied via keyletters, is recorded as part of the entry for the delta being created, and applies to *all* SCCS files processed by the same invocation of *delta*. This implies that if *delta* is invoked with more than one file argument, and the first file named has a **v** flag, all files named must have this flag. Similarly, if the first file named does not have this flag, then none of the files named may have it. Any file that does not conform to these rules is not processed.

When processing is complete, *delta* outputs (on the standard output) the SID of the created delta (obtained from the *p-file* entry) and the counts of lines inserted, deleted, and left unchanged by the delta. Thus, a typical output might be:

```
1.4
1.4 inserted
7 deleted
345 unchanged
```

It is possible that the counts of lines reported as inserted, deleted, or unchanged by *delta* do not agree with the user's perception of the changes applied to the *g-file*. The reason for this is that there usually are a number of ways to describe a set of such changes, especially if lines are moved around in the *g-file*, and *delta* is likely to find a description that differs from the user's perception. However, the *total* number of lines of the new delta (the number inserted plus the number left unchanged) should agree with the number of lines in the edited *g-file*.

If, in the process of making a delta, *delta* finds no ID keywords in the edited *g-file*, the message:

No id keywords (cm7)

is issued after the prompts for commentary, but before any other output. This indicates that any ID keywords that may have existed in the SCCS file have been replaced by their values, or deleted during the editing process. This could be caused by creating a delta from a *g-file* that was created by a *get* without the **-e** keyletter (recall that ID keywords are replaced by *get* in that case), or by accidentally deleting or changing the ID keywords during the editing of the *g-file*. Another possibility is that the file may never have had any ID keywords. In any case, it is left up to the user to determine what remedial action is necessary, but the delta is made, unless there is an **i** flag in the SCCS file, indicating that this should be treated as a fatal error. In this last case, the delta is not created.

After processing of an SCCS file is complete, the corresponding *p-file* entry is removed from the *p-file*<sup>7</sup>. If there is only *one* entry in the *p-file*, then the *p-file* itself is removed.

In addition, *delta* removes the edited *g-file*, unless the **-n** keyletter is specified. Thus:

*delta -n s.abc*

will keep the *g-file* upon completion of processing.

The **-s** ("silent") keyletter suppresses all output that is normally directed to the standard output, other than the prompts "comments?" and "MRs?". Thus, use of the **-s** keyletter together with the **-y** keyletter (and possibly, the **-m** keyletter) causes *delta* neither to read the standard input nor to write the standard output.

The differences between the *g-file* and the *d-file* (see above), which constitute the delta, may be printed on the standard output by using the **-p** keyletter. The format of this output is similar to that produced by *diff(1)*.

---

<sup>7</sup> All updates to the *p-file* are made to a temporary copy, the *q-file*, whose use is similar to the use of the *x-file*, which is described in Section 4 above

## SCCS User's Guide

### 5.3 admin

The **admin** command is used to *administer* SCCS files, that is, to create new SCCS files and to change parameters of existing ones. When an SCCS file is created, its parameters are initialized by use of keyletters or are assigned default values if no keyletters are supplied. The same keyletters are used to change the parameters of existing files.

Two keyletters are supplied for use in conjunction with detecting and correcting "corrupted" SCCS files, and are discussed in Section 6.3 below.

Newly-created SCCS files are given mode 444 (read-only) and are owned by the effective user. Only a user with write permission in the directory containing the SCCS file may use the **admin** command upon that file.

#### 5.3.1 Creation of SCCS Files

An SCCS file may be created by executing the command:

```
admin -ifirst s.abc
```

in which the value ("first") of the **-i** keyletter specifies the name of a file from which the text of the *initial delta* of the SCCS file "s.abc" is to be taken. Omission of the value of the **-i** keyletter indicates that **admin** is to read the standard input for the text of the initial delta. Thus the command:

```
admin -i s.abc < first
```

is equivalent to the previous example. If the text of the initial delta does not contain ID keywords, the message:

```
No id keywords (cm7)
```

is issued by **admin** as a warning. However, if the same invocation of the command also sets the **i** flag (not to be confused with the **-i** keyletter), the message is treated as an error and the SCCS file is not created. Only one SCCS file may be created at a time using the **-i** keyletter.

When an SCCS file is created, the *release* number assigned to its first delta is normally "1", and its *level* number is always "1". Thus, the first delta of an SCCS file is normally "1.1". The **-r** keyletter is used to specify the release number to be assigned to the first delta. Thus:

```
admin -ifirst -r3 s.abc
```

indicates that the first delta should be named "3.1" rather than "1.1". Because this keyletter is only meaningful in creating the first delta, its use is only permitted with the **-i** keyletter.

#### 5.3.2 Inserting Commentary for the Initial Delta

When an SCCS file is created, the user may choose to supply commentary stating the reason for creation of the file. This is done by supplying comments (-y keyletter) and/or MR numbers<sup>8</sup> (-m keyletter) in exactly the same manner as for *delta*. If comments (-y keyletter) are omitted, a comment line of the form:

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

is automatically generated.

If it is desired to supply MR numbers (-m keyletter), the **v** flag must also be set (using the **-f** keyletter described below). The **v** simply determines whether or not MR numbers must be supplied when using any SCCS command that modifies a *delta commentary* (see *sccsfile(5)*) in the SCCS file. Thus:

```
admin -ifirst -mmrnum1 -fv s.abc
```

Note that the **-y** and **-m** keyletters are only effective if a new SCCS file is being created.

---

8 The creation of an SCCS file may sometimes be the direct result of an MR.

**SCCS User's Guide****5.3.3 Initialization and Modification of SCCS File Parameters**

The portion of the SCCS file reserved for *descriptive text* (see Section 6.2) may be initialized or changed through the use of the -t keyletter. The descriptive text is intended as a summary of the contents and purpose of the SCCS file, although its contents may be arbitrary, and it may be arbitrarily long.

When an SCCS file is being created and the -t keyletter is supplied, it must be followed by the name of a file from which the descriptive text is to be taken. For example, the command:

```
admin -ifirst -tdesc s.abc
```

specifies that the descriptive text is to be taken from the file "desc".

When processing an *existing* SCCS file, the -t keyletter specifies that the descriptive text (if any) currently in the file is to be *replaced* with the text in the named file. Thus:

```
admin -tdesc s.abc
```

specifies that the descriptive text of the SCCS file is to be replaced by the contents of "desc"; omission of the file name after the -t keyletter as in:

```
admin -t s.abc
```

causes the *removal* of the descriptive text from the SCCS file.

The *flags* (see Section 6.2) of an SCCS file may be initialized and changed, or deleted through the use of the -f and -d keyletters, respectively. The flags of an SCCS file are used to direct certain actions of the various commands. See *admin(1)* for a description of all the flags. For example, the i flag specifies that the warning message stating there are no ID keywords contained in the SCCS file should be treated as an error, and the d (default SID) flag specifies the default version of the SCCS file to be retrieved by the *get* command. The -f keyletter is used to set a flag and, possibly, to set its value. For example:

```
admin -ifirst -fi -fmodname s.abc
```

sets the i flag and the m (module name) flag. The value "modname" specified for the m flag is the value that the *get* command will use to replace the %M% ID keyword. (In the absence of the m flag, the name of the g-file is used as the replacement for the %M% ID keyword.) Note that several -f keyletters may be supplied on a single invocation of *admin*, and that -f keyletters may be supplied whether the command is creating a new SCCS file or processing an existing one.

The -d keyletter is used to delete a flag from an SCCS file, and may only be specified when processing an existing file. As an example, the command:

```
admin -dm s.abc
```

removes the m flag from the SCCS file. Several -d keyletters may be supplied on a single invocation of *admin*, and may be intermixed with -f keyletters.

SCCS files contain a list (*user list*) of login names and/or group IDs of users who are allowed to create deltas (see Sections 5.1.3 and 6.2). This list is empty by default, which implies that *anyone* may create deltas. To add login names and/or groups IDs to the list, the -a keyletter is used. For example:

```
admin -axyz -awqi -a1234 s.abc
```

adds the login names "xyz" and "wqi" and the group ID "1234" to the list. The -a keyletter may be used whether *admin* is creating a new SCCS file or processing an existing one, and may appear several times. The -e keyletter is used in an analogous manner if one wishes to remove ("erase") login names or group IDs from the list.

## SCCS User's Guide

### 5.4 prs

*Prs* is used to print on the standard output all or parts of an SCCS file (see Section 6.2) in a format, called the output *data specification*, supplied by the user via the -d keyletter. The data specification is a string consisting of SCCS file *data keywords*<sup>9</sup> interspersed with optional user text.

Data keywords are replaced by appropriate values according to their definitions. For example:

:I:

is defined as the data keyword that is replaced by the SID of a specified delta. Similarly, :F: is defined as the data keyword for the SCCS file name currently being processed, and :C: is defined as the comment line associated with a specified delta. All parts of an SCCS file have an associated data keyword. For a complete list of the data keywords, see *prs(1)*.

There is no limit to the number of times a data keyword may appear in a data specification. Thus, for example:

*prs -d":I: this is the top delta for :F: :I:" s.abc*

may produce on the standard output:

2.1 this is the top delta for s.abc 2.1

Information may be obtained from a single delta by specifying the SID of that delta using the -r keyletter. For example:

*prs -d":F: :I: comment line is::C:" -r1.4 s.abc*

may produce the following output:

s.abc: 1.4 comment line is: THIS IS A COMMENT

If the -r keyletter is *not* specified, the value of the SID defaults to the most recently created delta.

In addition, information from a *range* of deltas may be obtained by specifying the -l or -e keyletters. The -e keyletter substitutes data keywords for the SID designated via the -r keyletter and all deltas created *earlier*. The -l keyletter substitutes data keywords for the SID designated via the -r keyletter and all deltas created *later*. Thus, the command:

*prs -d:I: -r1.4 -e s.abc*

may output:

1.4  
1.3  
1.2.1.1  
1.2  
1.1

and the command:

*prs -d:I: -r1.4 -l s.abc*

may produce:

3.1  
3.2  
3.1  
2.2.1.1  
2.2  
2.1  
1.4

Substitution of data keywords for *all* deltas of the SCCS file may be obtained by specifying both the -e and -l keyletters.

<sup>9</sup> Not to be confused with *get ID keywords*

**SCCS User's Guide****5.5 help**

The **help** command prints explanations of SCCS commands and of messages that these commands may print. Arguments to **help**, zero or more of which may be supplied, are simply the names of SCCS commands or the code numbers that appear in parentheses after SCCS messages. If no argument is given, **help** prompts for one. **Help** has no concept of *keyletter* arguments or *file* arguments. Explanatory information related to an argument, if it exists, is printed on the standard output. If no information is found, an error message is printed. Note that each argument is processed independently, and an error resulting from one argument will *not* terminate the processing of the other arguments.

Explanatory information related to a command is a synopsis of the command. For example:

```
help ge5 rmdel
```

produces:

```
ge5:  
  "nonexistent sid"  
  The specified sid does not exist in the  
  given file.  
  Check for typos.  
  
rmdel:  
  rmdel -rSID name ...
```

**5.6 rmdel**

The **rmdel** command is provided to allow *removal* of a delta from an SCCS file, though its use should be reserved for those cases in which incorrect, global changes were made a part of the delta to be removed.

The delta to be removed must be a "leaf" delta. That is, it must be the latest (most recently created) delta on its branch or on the trunk of the SCCS file tree. In Figure 3, only deltas 1.2.1.2, 1.3.2.2, and 2.2 can be removed and so on.

To be allowed to remove a delta, the effective user must have write permission in the directory containing the SCCS file. In addition, the real user must either be the one who created the delta being removed, or be the owner of the SCCS file and its directory.

The **-r** keyletter, which is mandatory, is used to specify the *complete* SID of the delta to be removed (i.e., it must have two components for a trunk delta, and four components for a branch delta). Thus:

```
rmdel -r2.3 s.abc
```

specifies the removal of (trunk) delta "2.3" of the SCCS file. Before removal of the delta, **rmdel** checks that the *release number* (R) of the given SID satisfies the relation:

*floor*  $\leq$  R  $\leq$  *ceiling*

**Rmdel** also checks that the SID specified is *not* that of a version for which a *get* for editing has been executed and whose associated *delta* has not yet been made. In addition, the login name or group ID of the user must appear in the file's *user list*, or the *user list* must be empty. Also, the release specified can not be *locked* against editing (i.e., if the l flag is set (see *admin(l)*), the release specified *must* not be contained in the list). If these conditions are not satisfied, processing is terminated, and the delta is not removed.

After the specified delta has been removed, its type indicator in the *delta table* of the SCCS file (see Section 6.2) is changed from "D" (for "delta") to "R" (for "removed").

## SCCS User's Guide

### 5.7 cdc

The *cdc* command is used to *change* a delta's commentary that was supplied when that delta was created. Its invocation is analogous to that of the *rmdel* command, except that the delta to be processed is *not* required to be a leaf delta. For example:

```
cdc -r3.4 s.abc
```

specifies that the commentary of delta "3.4" of the SCCS file is to be changed.

The *new* commentary is solicited by *cdc* in the same manner as that of *delta*. The old commentary associated with the specified delta is kept, but it is preceded by a comment line indicating that it has been changed (i.e., superseded), and the new commentary is entered ahead of this comment line. The "inserted" comment line records the login name of the user executing *cdc* and the time of its execution.

*Cdc* also allows for the deletion of selected MR numbers associated with the specified delta. This is specified by preceding the selected MR numbers by the character "!". Thus:

```
cdc -r1.4 s.abc  
MRs? mrnum3 !mrnum1  
comments? deleted wrong MR number and inserted correct MR number
```

inserts "mrnum3" and deletes "mrnum1" for delta 1.4.

### 5.8 what

The *what* command is used to find identifying information within *any* file whose name is given as an argument to *what*. Directory names and a name of "-" (a long minus sign) are not treated specially, as they are by other SCCS commands, and no *keyletters* are accepted by the command.

*What* searches the given file(s) for all occurrences of the string "@(#)", which is the replacement for the %Z% ID keyword (see *get(1)*), and prints (on the standard output) what follows that string until the first double quote (""), greater than (>), backslash (\), newline, or (non printing) NUL character. Thus, for example, if the SCCS file "s.prog.c" (which is a C program), contains the following line (the %M% and %I% ID keywords were defined in Section 5.1.1):

```
char id[] "%Z%%M%%I%";
```

and then the command:

```
get -r3.4 s.prog.c
```

is executed, and finally the resulting *g-file* is compiled to produce "prog.o" and "a.out", then the command:

```
what prog.c prog.o a.out
```

produces:

```
prog.c:  
    prog.c:3.4  
prog.o:  
    prog.c:3.4  
a.out:  
    prog.c:3.4
```

The string searched for by *what* need not be inserted via an ID keyword of *get*; it may be inserted in any convenient manner.

**SCCS User's Guide****5.9 sccsdiff**

The *sccsdiff* command determines (and prints on the standard output) the differences between two specified versions of one or more SCCS files. The versions to be compared are specified by using the -r keyletter, whose format is the same as for the *get* command. The two versions *must* be specified as the first two arguments to this command in the order in which they were created, ie., the older version is specified first. Any following keyletters are interpreted as arguments to the *pr(1)* command (which actually prints the differences) and must appear before any file names. SCCS files to be processed are named last. Directory names and a name of “-” (a long minus sign) are *not* acceptable to *sccsdiff*.

The differences are printed in the form generated by *diff(1)*. The following is an example of the invocation of *sccsdiff*:

```
sccsdiff -r3.4 -r5.6 s.abc
```

**5.10 comb**

*Comb* generates a *shell procedure* (see *sh(1)*) which attempts to reconstruct the named SCCS files so that the reconstructed files are smaller than the originals. The generated shell procedure is written on the standard output.

Named SCCS files are reconstructed by discarding unwanted deltas and combining specified other deltas. The intended use is for those SCCS files that contain deltas that are so old that they are no longer useful. It is *not* recommended that *comb* be used as a matter of routine; its use should be restricted to a *very small number* of times in the life of an SCCS file.

In the absence of any keyletters, *comb* preserve only leaf deltas and the minimum number of ancestor deltas necessary to preserve the “shape” of the SCCS file tree. The effect of this is to eliminate “middle” deltas on the trunk and on all branches of the tree. Thus, in Figure 3, deltas 1.2, 1.3.2.1, and 2.1 would be eliminated. Some of the keyletters are summarized as follows:

The -p keyletter specifies the oldest delta that is to be preserved in the reconstruction. All older deltas are discarded.

The -c keyletter specifies a *list* (see *get(1)* for the syntax of such a list) of deltas to be preserved. All other deltas are discarded.

The -s keyletter causes the generation of a shell procedure, which, when run, produces *only* a report summarizing the percentage space (if any) to be saved by reconstructing each named SCCS file. It is recommended that *comb* be run with this keyletter (in addition to any others desired) *before* any actual reconstructions.

It should be noted that the shell procedure generated by *comb* is *not* guaranteed to save any space. In fact, it is possible for the reconstructed file to be *larger* than the original. Note, too, that the shape of the SCCS file tree may be altered by the reconstruction process.

**5.11 val**

*Val* is used to determine if a file is an SCCS file meeting the characteristics specified by an optional list of keyletter arguments. Any characteristics not met are considered errors.

*Val* checks for the existence of a particular delta when the SID for that delta is explicitly specified via the -r keyletter. The string following the -y or -m keyletter is used to check the value set by the t or m flag respectively (see *admin(1)* for a description of the flags).

*val* treats the special arguments “-” differently from other SCCS commands (see Section 4). This argument allows *val* to read the argument list from the standard input as opposed to obtaining it from the command line. The standard input is read until end-of-file. This capability allows for one invocation of *val* with different values for the keyletter and file arguments. For example:

```
val -
-yC -mabc s.abc
-mxyz -yp11 s.xyz
```

## SCCS User's Guide

first checks if file "s.abc" has a value "c" for its *type* flag and value "abc" for the *module name* flag. Once processing of the first file is completed, *val* then processes the remaining files, in this case "s.xyz", to determine if they meet the characteristics specified by the keyletter arguments associated with them.

*Val* returns an 8-bit code which is a disjunction of the possible errors detected. That is, each bit set indicates the occurrence of a specific error (see *val*[1] for a description of the possible errors and their codes). In addition, an appropriate diagnostic is printed unless suppressed by the -s keyletter. A return code of "0" indicates all named files met the characteristics specified.

## 6. SCCS FILES

This section discusses several topics that must be considered before extensive use is made of SCCS. These topics deal with the protection mechanisms relied upon by SCCS, the format of SCCS files, and the recommended procedures for auditing SCCS files.

### 6.1 Protection

SCCS relies on the capabilities of the operating system for most of the protection mechanisms required to prevent unauthorized changes to SCCS files (i.e., changes made by non-SCCS commands). The only protection features provided directly by SCCS are the *release lock* flag, the *release floor* and *ceiling* flags, and the *user list* (see Section 5.1.3).

New SCCS files created by the *admin* command are given mode 444 (read only). It is recommended that this mode *not be* changed, as it prevents any direct modification of the files by non-SCCS commands. It is further recommended that the directories containing SCCS files be given mode 755, which allows only the *owner* of the directory to modify its contents.

SCCS files should be kept in directories that contain only SCCS files and any temporary files created by SCCS commands. This simplifies protection and auditing of SCCS files (see Section 6.3). The contents of directories should correspond to convenient logical groupings, e.g., sub-systems of a large project.

SCCS files must have only *one link* (name). The reason for this is that those commands that modify SCCS files do so by creating a temporary copy of the file (called the *x-file*, see Section 4) and, upon completion of processing, remove the old file and rename the *x-file*. If the old file has more than one link, removing it and renaming the *x-file* would break the link. Rather than process such files, SCCS commands produce an error message. All SCCS files *must* have names that begin with "s.".

When only one user uses SCCS, the real and effective user IDs are the same, and that user ID owns the directories containing SCCS files<sup>10</sup>. Therefore, SCCS may be used directly without any preliminary preparation.

However, in those situations in which several users with unique user IDs are assigned responsibility for one SCCS file (for example, in large software development projects), one user (equivalently, one user ID) must be chosen as the "owner" of the SCCS files and be the one who will "administer" them (e.g., by using the *admin* command). This user is termed the *SCCS administrator* for that project. Because other users of SCCS do not have the same privileges and permissions as the SCCS administrator, they are not able to execute directly those commands that require write permission in the directory containing the SCCS files. Therefore, a project-dependent program is required to provide an interface to the *get*, *delta*, and, if desired, *rmdel* and *cdc* commands.

10 Previously, the Operating System under which SCCS executed allowed for only 256 unique user IDs. This presented the situation in which several users needed to share user IDs (and thus shared identical file permissions). The Operating System currently in use (Version 7 of UNIX allows for 55,536 unique user IDs, and it is recommended that each user have a unique user ID.

## SCCS User's Guide

The interface program must be owned by the SCCS administrator, and must have the *set user ID on execution bit on* (see *chmod(1)*), so that the effective user ID is the user ID of the administrator. This program's function is to invoke the desired SCCS command and to cause it to *inherit* the privileges of the interface program for the duration of that command's execution. In this manner, the owner of an SCCS file can modify it at will.

Other users whose *login names* or *group IDs* are in the *user list* for that file (but who are *not* its owners) are given the necessary permissions only for the duration of the execution of the interface program, and are thus able to modify the SCCS files only through the use of *delta* and, possibly, *rmdel* and *cdc*. The project-dependent interface program, as its name implies, must be custom-built for each project.

### 6.2 Format

SCCS files are composed of lines of ASCII text<sup>11</sup> arranged in six parts, as follows:

Checksum	A line containing the "logical" sum of all the characters of the file ( <i>not</i> including this checksum itself).
Delta Table	Information about each delta, such as its type, its SID, date and time of creation, and commentary.
User Names	List of login names and/or group IDs of users who are allowed to modify the file by adding or removing deltas.
Flags	Indicators that control certain actions of various SCCS commands.
Descriptive Text	Arbitrary text provided by the user; usually a summary of the contents and purpose of the file.
Body	Actual text that is being administered by SCCS, intermixed with internal SCCS control lines.

Detailed information about the contents of the various sections of the file may be found in *sccsfile(5)*; the *checksum* is the only portion of the file which is of interest below.

It is important to note that because SCCS files are ASCII files, they may be processed by various commands, such as *ed(1)*, *grep(1)*, and *cat(1)*. This is very convenient in those instances in which an SCCS file must be modified manually (e.g., when the time and date of a delta was recorded incorrectly because the system clock was set incorrectly), or when it is desired to simply "look" at the file.

Note:*Extreme care should be exercised when modifying SCCS files with non-SCCS commands.*

### 6.3 Auditing

On rare occasions, perhaps due to an operating system or hardware malfunction, an SCCS file, or portions of it (i.e., one or more "blocks") can be destroyed. SCCS commands (like most commands) issue an error message when a file does not exist. In addition, SCCS commands use the *checksum* stored in the SCCS file to determine whether a file has been *corrupted* since it was last accessed (possibly by having lost one or more blocks, or by having been modified with, for example, *ed(1)*). No SCCS command will process a corrupted SCCS file except the *admin* command with the -h or -z keyletters, as described below.

It is recommended that SCCS files be audited (checked) for possible corruptions on a regular basis. The simplest and fastest way to perform audit is to execute the *admin* command with the -h keyletter on all SCCS files:

```
admin -h s.file1 s.file2
      or
admin -h directory1 directory2 ...
```

---

<sup>11</sup> Previous versions of SCCS up to and including Version 3 used non-ASCII files. Therefore, files created by earlier versions of SCCS are incompatible with the current version of SCCS.

## SCCS User's Guide

If the new checksum of any file is not equal to the checksum in the first line of that file, the message:

corrupted file (co6)

is produced for that file. This process continues until all the files have been examined. When examining directories (as in the second example above), the process just described will not detect *missing* files.

A simple way to detect whether *any* files are missing from a directory is to periodically execute the *ls(1)* command on that directory, and compare the outputs of the most current and the previous executions. Any file whose name appears in the previous output but not in the current one has been removed by some means.

Whenever a file has been corrupted, the manner in which the file is restored depends upon the extent of the corruption. If damage is extensive, the best solution is to contact the local operations group to request a restoral of the file from a backup copy. In the case of minor damage, repair through use of the editor *ed(1)* may be possible. In the latter case, after such repair, the following command must be executed:

admin -z s.file

The purpose of this is to recompute the checksum the bring it into agreement with the actual contents of the file. After this command is executed on a file, any corruption which may have existed in that file will no longer be detectable.

## REFERENCES

- [1] Bell Laboratories, *Documents for Use with the PWB Time-Sharing System*

**SCCS User's Guide****ADDENDUM**

The following changes to the Source Code Control System are effective with the UNIX™ System III release.

**1. Modified commands**

Three SCCS commands have been modified:

1. comb
2. get
3. sccsdiff

Modifications to each of these commands are described below.

**1.1 comb**

- enhancement

*Comb* generates a shell procedure that, when executed, will (hopefully) reduce the size of an SCCS file. Because of temporary file naming conventions, two or more *comb* generated shell procedures could not be executed concurrently. Temporary files are now uniquely named so that simultaneous execution is possible.

**1.2 get**

- enhancement

Previously the -i and -x keyletters (for forced inclusion or exclusion of deltas to produce the generated file) would imply the -k keyletter. That is, the generated file would be created with mode 644 and identification keyword replacement would be suppressed. The -i and -x keyletters no longer imply the -k keyletter.

- coding error correction

Under certain circumstances, temporary files that should only have existed for the duration of the execution of *get* would not be removed when *get* terminated. Temporary files are now properly removed.

**1.3 sccsdiff**

- new capability

A new keyletter (-s), which takes a numeric argument, allows the user to specify the file segmentation size that *bdiff(1)* (used by *sccsdiff*) will pass to *diff(1)*. This can be useful when a high system load causes *diff* to fail due to lack of space, etc.

- change

The output of *sccsdiff* is no longer piped through *pr(1)* by default. A new keyletter (-p) specifies that the output is to be piped through *pr* but arguments can not be passed to *pr* as was the previous case. This alleviates *sccsdiff* knowing anything about *pr*.

**2. New Commands**

Two new commands have been added to SCCS:

*sact* print current SCCS file editing activity

*unget* undo the effect of a previous *get(1)* for editing of an SCCS file.

The manual entries for these commands are provided in the UNIX™ System III User's Manual.

## SCCS Interface Program

### Function and Use of an SCCS Interface Program

L.E.Bonanni

A. Guyton (4/1/80 revision)

Bell Laboratories

Piscataway, New Jersey 08854

#### ABSTRACT

This memorandum discusses the use of a Source Code Control System Interface Program to allow more than one user to use SCCS commands upon the same set of files.

### 1. INTRODUCTION

In order to permit UNIX<sup>†</sup> users with different user identification numbers (user IDs) to use SCCS commands upon the same files, an SCCS Interface program is provided to temporarily grant the necessary file access permissions to these users. This memorandum discusses the creation and use of such an interface program. This memorandum replaces an earlier version dated March 1, 1978.

### 2. FUNCTION

When only one user uses SCCS, the real and effective user IDs are the same, and that user ID owns the directories containing SCCS files. However, there are situations (for example, in large software development projects) in which it is practical to allow more than one user to make changes to the same set of SCCS files. In these cases, one user must be chosen as the owner of the SCCS files and be the one who will administer them (e.g., by using the *admin* command). This user is termed the SCCS administrator for that project. Since other users of SCCS do not have the same privileges and permissions as the SCCS administrator, they are not able to execute directly those commands that require write permission in the directory containing the SCCS files. Therefore, a project-dependent program is required to provide an interface to the *get*, *delta*, and, if desired, *rmdel*, *cdc*, and *unget* commands.<sup>1</sup>

The interface program must be owned by the SCCS administrator, must be executable by non-owners, and must have *set user ID on execution* bit on (see *chmod(1)*<sup>2</sup>), so that, when executed, the *effective* user ID is the user ID of administrator. This program's function is to invoke the desired SCCS command and to cause it to *inherit* the privileges of the SCCS administrator for the duration of that command's execution. In this manner, the owner of an SCCS file (the administrator) can modify it at will. Other users whose *login* names are in the *user list*<sup>3</sup> for that file (but who are *not* its owners) are given the necessary permissions only for the duration of the execution of the interface program, and are thus able to modify the SCCS files only through the use of *delta* and, possibly, *rmdel* and *cdc*.

### 3. A BASIC PROGRAM

When a UNIX program is executed it is passed (as argument 0) the *name* by which it is invoked, followed by any additional user-supplied arguments. Thus, if a program is given a number of *links* (names), it may alter its processing depending upon which link is used to invoke it. The mechanism is used by an SCCS interface program to determine which SCCS command it should subsequently invoke (see *exec(2)*).

<sup>†</sup> UNIX is a trademark of Bell Laboratories

<sup>1</sup> Other SCCS commands either do not require write permission in the directory containing SCCS files or are (generally) reserved for use only by the administrator.

<sup>2</sup> All references of the form *name(N)* refer to item *name* in section *N* of the *UPM(1)*

<sup>3</sup> This is the list of login names of users who are allowed to modify an SCCS file by adding or removing deltas. The login names are specified using the *admin(1)* command.

## SCCS Interface Program

A generic interface program (*inter.c*, written in C) is shown in *Attachment I*. Note the reference to the (un supplied) function *filearg*. This is intended to demonstrate that the interface program may also be used a pre-processor to SCCS commands. For example, function *filearg* could be used to modify file arguments to be passed to the SCCS command by supplying the *full path name* of a file, thus avoiding extra typing by the user. Also, the program could supply any additional (default) keyletter arguments desired.

### 4. LINKING AND USE

In general, the following demonstrates the steps to be performed by the SCCS administrator to create the SCCS interface program. It is assumed, for the purposes of the discussion, that the interface program *inter.c* resides in directory */x1/xyz/sccs*. Thus, the command sequence:

```
cd /x1/xyz/sccs  
cc ... inter.c -o inter ...
```

compiles *inter.c* to produce the executable module *inter* (... represents arguments that may also be required). The proper mode and the *set user ID on execution* bit are set by executing:

```
chmod 4755 inter
```

Finally new links are created, by (for example):<sup>4</sup>

```
ln inter get  
ln inter delta  
ln inter rmdel
```

Subsequently, *any* user whose shell parameter PATH (see *sh(1)*) specifies that directory */x1/xyz/sccs* is to be searched first for executable commands, may execute, for example:

```
get -e /x1/xyz/sccs/s.abc
```

from any directory to invoke the interface program (via its link *get*). The interface program then executes */usr/bin/get* (the actual SCCS *get* command) upon the named file. As previously mentioned, the interface program could be used to supply the pathname */x1/xyz/sccs*, so that the user would only have to specify:

```
get -e s.abc
```

to achieve the same results.

### 5. CONCLUSION

An SCCS interface program is used to permit users having different user IDs to use SCCS commands upon the same files. Although this is its primary purpose, such a program may also be used as a pre-processor to SCCS commands since it can perform operations upon its arguments.

---

<sup>4</sup> The names of the links may be arbitrary, provided the interface program is able to determine from them the names of SCCS commands to be invoked.

**SCCS Interface Program****Attachment I****SCCS Interface Program inter.c**

```
main(argc, argv)
int argc;
char*argv[];
{
    register int i;
    char cmdstr[LENGTH]
    /* Process file arguments (those that don't begin with "-"). */
    for (i = 1;i < argc; i++)
        if (argv[i][0] != '-')
            argv[i] = filearg(argv[i]);
    /* Get "simple name" of name used to invoke this program */
    /* (i.e., strip off directory-name prefix, if any). */
    argv[0] = sname(argv[0]);
    /* Invoke actual SCCS command, passing arguments */
    sprintf(cmdstr, "/usr/bin/%s", argv[0]);
    execv(cmdstr, argv);
}
```

*January 1981*

**FSCK****FSCK-The UNIX/TS File System Check Program**  
*T.J.Kowalski*

Bell Laboratories  
Murray Hill, New Jersey 07974

**1. INTRODUCTION**

When a UNIX/TS operating system is brought up, a consistency check of the file system should always be performed. This precautionary measure helps to insure a reliable environment for file storage on disc. If an inconsistency is discovered, corrective action must be taken. No changes are made to any file system by *fsck* without prior operator approval.

The purpose of this memo is to dispel the mystique surrounding file system inconsistencies. It first describes the updating of the file system (the calm before the storm) and then describes file system corruption (the storm). Finally, the set of heuristically sound corrective actions used by *fsck* (the Coast Guard to the rescue) is presented.

**2. UPDATE OF THE FILE SYSTEM**

Every working day hundreds of files are created, modified, and removed. Every time a file is modified, the UNIX operating system performs a series of file system updates. These updates, when written on disc, yield a consistent file system. To understand what happens in the event of a permanent interruption in this sequence, it is important to understand the order in which the update requests were probably being honored. Knowing which pieces of information were probably written to the file system first, heuristic procedures can be developed to repair a corrupted file system.

There are five types of file system updates. These involve the superblock, i-nodes, indirect blocks, data blocks (directories and files), and free-list blocks.

**2.1 Superblock**

The superblock contains information about the size of the file system, the size of the i-node list, part of the free-block list, the count of free blocks, the count of free i-nodes, and part of the free i-node list.

The super-block of a mounted file system (the root file system is always mounted) is written to the file system whenever the file is unmounted or a *sync* command is issued.

**2.2 I-nodes**

An i-node contains information about the type of i-node (directory, data, or special), the number of directory entries linked to the i-node, the list of blocks claimed by the i-node, and the size of the i-node.

An i-node is written to the file system upon closure<sup>1</sup> of the file associated with the i-node.

**2.3 Indirect Blocks**

There are three types of indirect blocks: single-indirect, double-indirect and triple-indirect. A single-indirect block contains a list of some of the block numbers claimed by an i-node. Each one of the 128 entries in an indirect block is a data-block number. A double-indirect block contains a list of single-indirect block numbers. A triple-indirect block contains a list of double-indirect block numbers.

Indirect blocks are written to the file system whenever they have been modified and released<sup>2</sup> by the operating system.

---

<sup>1</sup> All in core blocks are also written to the file system upon issue of a *sync* system call

<sup>2</sup> More precisely, they are queued for eventual writing. Physical I/O is deferred until the buffer is needed by UNIX/TS or a *sync* command is issued

**FSCK****2.4 Data Blocks**

A data block may contain file information or directory entries. Each directory consists of a file name and an i-node number.

Data blocks are written to the file system whenever they have been modified and released by the operating system.

**2.5 First Free-List Block**

The superblock contains the first free-list block. The free-list blocks are a list of all blocks that are not allocated to the superblock, i-nodes, indirect blocks, or data blocks. Each free-list block contains a count of the number of entries in this free-list block, a pointer to the next free-list block, and a partial list of free blocks in the file system.

Free-list blocks are written to the file system whenever they have been modified and released by the operating system.

**3. CORRUPTION OF THE FILE SYSTEM**

A file system can become corrupted in a variety of ways. The most common of these ways are improper shutdown procedures and hardware failures.

**3.1 Improper System Shutdown and Startup**

File systems may become corrupted when proper shutdown procedures are not observed, e.g., forgetting to *sync* the system prior to halting the CPU, physically write-protecting a mounted file system, or taking a mounted file system off-line.

File systems may become further corrupted if proper startup procedures are not observed, e.g., not checking a file system for inconsistencies, and not repairing inconsistencies. Allowing a corrupted file system to be used (and, thus, to be modified further) can be disastrous.

**3.2 Hardware Failure**

Any pieces of hardware can fail at any time. Failure can be as subtle as a bad block on a disk pack, or as blatant as a non-functional disk-controller.

**4. DETECTION AND CORRECTION OF CORRUPTION**

A quiescent<sup>3</sup> file system may be checked for structural integrity by performing consistency checks on the redundant data intrinsic to a file system. The redundant data is either read from the file system or computed from other known values. A quiescent state is important during the checking of a file system because of the multi-pass nature of the *fsck* program.

When an inconsistency is discovered *fsck* reports the inconsistency for the operator to choose a corrective action.

Discussed in this section are how to discover inconsistencies and possible corrective actions for the superblock, the i-nodes, the indirect blocks, the data blocks containing directory entries, and the free-list blocks. These corrective actions can be performed interactively by the *fsck* command under control of the operator.

---

3 That is, unmounted and not being written on.

## FSCK

### 4.1 Super-Block

One of the most common corrupted items is the superblocks. The superblock is prone to corruption because every change to the file system's block or i-nodes modifies the superblock.

The superblock and its associated parts are most often corrupted when the computer is halted and the last command involving output to the file system was not a *sync* command.

The superblock can be checked for inconsistencies involving file-system size, i-node list size, free-block list, free-block count, and the free i-node count.

#### 4.1.1 File-System Size and I-node List Size

The file-system size must be larger than the number of blocks used by the superblocks and the number of blocks used by the list of i-nodes. The number of i-nodes must be less than 65,535. The file-system size and i-node list size are critical pieces of information to the *fsck* program. While there is no way to actually check these sizes, *fsck* can check for them being within reasonable bounds. All other checks of the file system depend on the correctness of these sizes.

#### 4.1.2 Free-Block List

The free-block list starts in the superblock and continues through the free-list blocks of the file system. Each free-list block can be checked for a list count out of range, for block numbers out of range, and for blocks already allocated within the file system. A check is made to see that all the blocks in the file system were found.

The first free-block list is in the superblock. *Fsck* checks the list count for a value of less than zero or greater than fifty. It also checks each block number for a value of less than the first data block in the file system or greater than the last block in the file system. Then it compares each block number to a list of already allocated blocks. If the free-list block pointer is non-zero, the next free-list block is read in and the process repeated.

When all the blocks have been accounted for, a check is made to see if the number of blocks used by the free-block list plus the number of blocks claimed by the i-nodes equals the total number of blocks in the file system.

If anything is wrong with the free-block list, then *fsck* may rebuild it, excluding all blocks in the list of allocated blocks.

#### 4.1.3 Free-Block Count

The superblock contains a count of the total number of free blocks within the file system. *Fsck* compares this count to the number of blocks it found free within the file system. If they don't agree, then *fsck* may replace the count in the superblock by the actual free-block count.

#### 4.1.4 Free I-node Count

The superblock contains a count of the total number of free i-nodes within the file system. *Fsck* compares this count to the number of i-nodes it found free within the file system. If they don't agree, then *fsck* may replace the count in the superblock by the actual free i-node count.

### 4.2 I-nodes

An individual i-node is not as likely to be corrupted as the superblock. However, because of the great number of active i-nodes, there is almost as likely a chance for corruption in the i-node list as in the superblock.

The list of i-nodes is checked sequentially starting with i-node 1 (there is no i-node 0) and going to the last i-node in the file system. Each i-node can be checked for inconsistencies involving format and type, link count, duplicate blocks, bad blocks, and i-node size.

**FSCK****4.2.1 Format and Type**

Each i-node contains a mode word. This mode word describes the type and state of the i-node. Inodes may be one of four types: regular i-node, directory i-node, special block i-node, and special character i-node. If an i-node is not one of these types, then the i-node has an illegal type. I-nodes may be found in one of three states: unallocated, allocated, and neither unallocated nor allocated. This last state indicates an incorrectly formatted i-node. An i-node can get in this state if bad data is written into the i-node list through, for example, a hardware failure. The only possible corrective action is for *fsck* is to clear the i-node.

**4.2.2 Link Count**

Contained in each i-node is a count of the total number of directory entries linked to the i-node.

*Fsck* verifies the link count of each i-node by traversing down the total directory structure, starting from the root directory, calculating an actual link count for each i-node.

If the stored link count is non-zero and the actual link count is zero, it means that no directory entry appears for the i-node. If the stored and actual link counts are non-zero and unequal, a directory entry may have been added or removed without the i-node being updated.

If the stored link count is non-zero and the actual link count is zero, *fsck* may link the disconnected file to the *lost + found* directory. If the stored and actual link counts are non-zero and unequal, *fsck* may replace the stored link count by the actual link count.

**4.2.3 Duplicate Blocks**

Contained in each i-node is a list or pointers to lists (indirect blocks) of all the blocks claimed by the i-node.

*Fsck* compares each block number claimed by an i-node to a list of already allocated blocks. If a block number is already claimed by another i-node, the block number is added to a list of duplicate blocks. Otherwise, the list of allocated blocks is updated to include the block number. If there are any duplicate blocks, *fsck* will make a partial second pass of the i-node list to find the i-node of the duplicated block, because without examining the files associated with these i-nodes for correct content, there is not enough information available to decide which i-node is corrupted and should be cleared. Most times, the i-node with the earlier modify time is incorrect, and should be cleared.

This condition can occur by using a file system with blocks claimed by both the free-block list and by other parts of the file system.

If there is a large number of duplicate blocks in an i-node, this may be due to an indirect block not being written to the file system.

*Fsck* will prompt the operator to clear both i-nodes.

**4.2.4 Bad Blocks**

Contained in each i-node is a list or pointer to lists of all the blocks claimed by the i-node.

*Fsck* checks each block number claimed by an i-node for a value lower than that of the first data block, or greater than the last block in the file system. If the block number is outside this range, the block number is a bad block number.

If there is a large number of bad blocks in an i-node, this may be due to an indirect block not being written to the file system.

*Fsck* will prompt the operator to clear both i-nodes.

## FSCK

### 4.2.5 Size checks

Each i-node contains a thirty-two bit (four-byte) size field. This size indicates the number of characters in the file associated with the i-node. This size can be checked for inconsistencies, e.g., directory sizes that are not a multiple of sixteen characters, or the number of blocks actually used not matching that indicated by the i-node size.

A directory i-node within the UNIX file system has the directory bit on in the i-node mode word. The directory i-node must be a multiple of sixteen because a directory contains sixteen bytes of information (two bytes for the i-node number and fourteen bytes for the file or directory name).

*Fsck* will warn of such directory misalignment. This is only a warning because not enough information can be gathered to correct the misalignment.

A rough check of the consistency of the size field of an i-node can be performed by computing from the size field the number of blocks that should be associated with the i-node and comparing it to the actual number of blocks claimed by the i-node.

*Fsck* calculates the number of blocks that there should be in an i-node by dividing the number of characters in an i-node by the number of characters per block (512) and rounding up. *Fsck* adds one block for each indirect block associated with the i-node. If the actual number of blocks does not match the computed number of blocks, *fsck* will warn of a possible file-size error. This is only a warning because UNIX/TS does not fill in blocks created in random order.

### 4.3 Indirect Blocks

Indirect blocks are owned by an i-node. Therefore, inconsistencies in indirect blocks directly affect the i-node that owns it.

Inconsistencies that can be checked are blocks already claimed by another i-node and block numbers outside the range of the file system.

For a discussion of detection and correction of the inconsistencies associated with indirect blocks, apply iteratively Sections 4.2.3 and 4.2.4 to each level of indirect blocks.

### 4.4 Data Blocks

The two types of data blocks are plain data blocks and directory data blocks. Plain data blocks contain the information stored in a file. Directory data blocks contain directory entries. *Fsck* does not attempt to check the validity of the contents of a plain data block.

Each directory data block can be checked for inconsistencies involving directory i-node numbers pointing to unallocated i-nodes, directory i-node numbers greater than the number of i-nodes in the file system, incorrect directory i-node numbers for “.” and “..”, and directories which are disconnected from the file system.

If a directory entry i-node number points to an unallocated i-node, then *fsck* may remove that directory entry. This condition probably occurred because the data blocks containing the directory entries were modified and written to the file system while the i-node was not yet written out.

If a directory entry i-node number is pointing beyond the end of the i-node list, *fsck* may remove that directory entry. This condition occurs if bad data is written into a directory data block.

The directory i-node number entry for “.” should be the first entry in the directory data block. Its value should be equal to the i-node number for the directory data block.

The directory i-node number entry for “..” should be the second entry in the directory data block. Its value should be equal to the i-node number for the parent of the directory entry (or the i-node number of the directory data block if the directory is the root directory).

If the directory i-node numbers are incorrect, *fsck* may replace them by the correct values.

## FSCK

*Fsck* checks the general connectivity of the file system. If directories are found not to be linked into the file system, *fsck* will link the directory back into the file system in the *lost + found* directory. This condition can be caused by i-nodes being written to the file system with the corresponding directory data blocks not being written to the file system.

### 4.5 Free-List Blocks

Free-list blocks are owned by the superblock. Therefore, inconsistencies in free-list blocks directly affect the superblock.

Inconsistencies that can be checked are a list count outside of range, block numbers outside of range, and blocks already associated with the file system.

For a discussion of detection and correction of the inconsistencies associated with free-list blocks see Section 4.1.2.

## ACKNOWLEDGEMENT

I would like to thank Larry A. Wehr for advice that lead to the first version of *fsck* and Rick B. Brandt for adapting *fsck* to UNIX/TS.

## REFERENCE

- [1] Ritchie, D. M., and Thompson, K., The UNIX Time-Sharing System, *The Bell System Technical Journal* 57, 6 (July-August 1978, Part2), pp. 1905-29.
- [2] Dolotta, T. A., and Olsson, S. B. eds., *UNIX/TS User's Manual, Edition 1.1* (January 1978).
- [3] Thompson, K., UNIX Implementation, *The Bell System Technical Journal* 57, 6 (July-August 1978, Part 2),pp. 1931-46.

## FSCK

### Appendix-FSCK ERROR CONDITIONS

#### 1. CONVENTIONS

*Fsck* is a multi-pass file system check program. Each file system pass invokes a different Phase of the *fsck* program. After the initial setup, *fsck* performs successive Phases over each file system, checking blocks and sizes, path-names, connectivity, reference counts, and the free-block list (possibly rebuilding it), and performs some cleanup.

When an inconsistency is detected, *fsck* reports the error condition to the operator. If a response is required, *fsck* prints a prompt message and waits for a response. This appendix explains the meaning of each error condition, the possible response, and the related error conditions.

The error conditions are organized by the *Phase* of the *fsck* program in which they can occur. The error conditions that may occur in more than one Phase will be discussed in initialization.

#### 2. INITIALIZATION

Before a file system check can be performed, certain tables have to be set up and certain files opened. This section concerns itself with the opening of files and the initialization of tables. This section lists error conditions resulting from command line options, memory requests, opening of files, status of files, file system size checks, and creation of the scratch file.

##### C option?

*C* is not a legal option to *fsck*, legal options are *-y*, *-n*, *-s*, *-S*, and *-t*. *Fsck* terminates on this error condition. See the *fsck(1M)* manual entry for further detail.

##### Bad -t option

The *-t* option is not followed by a file name. *Fsck* terminates on this error condition. See the *fsck(1M)* manual entry for further detail.

##### Invalid -s arguments, defaults assumed

The *-s* option is not suffixed by 3, 4, or blocks-per-cylinder:blocks-to-skip. *Fsck* assumes a default value of 400 blocks-per-cylinder and 9 blocks-to skip. See the *fsck(1M)* manual entry for more details.

##### incompatible options: -n and -s

It is not possible to salvage the free-block list without modifying the file system. *Fsck* terminates on this error condition. See the *fsck(1M)* manual for further detail.

##### Can't get memory

*Fsck*'s request for memory for its virtual memory tables failed. This should never happen. *Fsck* terminates on this error condition. See a guru.

##### Can't open checklist file: F

The default file system checklist file *F* (usually */etc/checklist*) can not be opened for reading. *Fsck* terminates on this error condition. Check access modes of *F*.

##### Can't stat root

*Fsck*'s request for statistics about the root directory "/" failed. This should never happen. *Fsck* terminates on this error condition. See a guru.

**FSCK****Can't stat F**

*Fsck's request for statistics about the file system F failed. It ignores this file system and continues checking the next file system given. Check access modes of F.*

**F is not a block or character device**

*You have given fsck a regular file name by mistake. It ignores this file system and continues checking the next file system given. Check file type of F.*

**Can't open F**

*The file system F can not be opened for reading. It ignores this file system and continues checking the next file system given. Check access modes of F.*

**Size check: fsize Z isize Y**

*More blocks are used for the i-node list Y than there are blocks in the file system X, or there are more than 65,535 i-nodes in the file system. It ignores this file system and continues checking the next file system given. See Section 4.1.1.*

**Can't create F**

*Fsck's request to create a scratch file F failed. It ignores this file system and continues checking the next file system given. Check access modes of F.*

**CAN NOT SEEK: BLX B (CONTINUE)**

*Fsck's request for moving to a specified block number B in the file system failed. This should never happen. See a guru.*

Possible responses to the **CONTINUE** prompt are:

YES      Attempt to continue to run the file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If the block was part of the virtual memory buffer cache, *fsck* will terminate with the message "Fatal I/O error".

NO      Terminate the program.

**CAN NOT READ: BLK B (CONTINUE)**

*Fsck's request for reading a specified block number B in the file system failed. This should never happen. See a guru.*

Possible responses to the **CONTINUE** prompt are:

YES      Attempt to continue to run the file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If the block was part of the virtual memory buffer cache, *fsck* will terminate with the message "Fatal I/O error".

NO      Terminate the program.

## FSCK

### CAN NOT WRITE: BLK B (CONTINUE)

*Fsck's* request for writing a specified block number **B** in the file system failed. The disk is write-protected. See a guru.

Possible responses to the **CONTINUE** prompt are:

**YES** Attempt to continue to run the file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If the block was part of the virtual memory buffer cache, *fsck* will terminate with the message "Fatal I/O error"

**NO** Terminate the program.

### 3. PHASE 1: CHECK BLOCKS AND SIZES

This phase concerns itself with the i-node list. This section lists error conditions resulting from checking i-node types, setting up the zero-link-count table, examining i-node block numbers for bad or duplicate blocks, checking i-node size, and checking i-node format.

### UNKNOWN FILE TYPE I = I (CLEAR)

The mode word of the i-node **I** indicates that the i-node is not a special character i-node, regular i-node, or directory i-node. See Section 4.2.1.

Possible responses to the **CLEAR** prompt are:

**YES** De-allocate i-node **I** by zeroing its contents. This will always invoke the **UNALLOCATED** error condition in Phase 2 for each directory pointing to this i-node.

**NO** Ignore this error condition.

### LINK COUNT TABLE OVERFLOW (CONTINUE)

An internal table for *fsck* containing allocated i-nodes with a link count of zero has no more room. Recompile *fsck* with a large value of **MAXLNCNT**.

Possible responses to the **CONTINUE** prompt are:

**YES** Continue with the program. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If another allocated i-node with a zero link count is found, this error condition is repeated.

**NO** Terminate the program.

### B BAD I=I

I-node **I** contains block number **B** with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system. This error condition may invoke the **EXCESSIVE BAD BLKS** error condition in Phase 1 if i-node **I** has too many block numbers outside the file system range. This error condition will always invoke the **BAD/DUP** error condition in Phase 2 and Phase 4. See Section 4.2.4.

**FSCK****EXCESSIVE BAD BLKS I=I (CONTINUE)**

There is more than a tolerable number (usually 10) of blocks with a number lower than the number of the first data block in the file system or greater than the number of last block in the file system associated with i-node I. See section 4.2.4.

Possible responses to the **CONTINUE** prompt are:

YES Ignore the rest of the blocks in this i-node and continue checking with the next i-nodes in the file system. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system.

NO Terminate the program.

**B DUP I=I**

I-node I contains block number B which is already claimed by another i-node. This error condition may invoke the **EXCESS DUP BLKS** error condition in Phase 1 if i-node I has too many block numbers claimed by other i-nodes. This error condition will always invoke Phase 1b and the **BAD/DUP** error condition in Phase 2 and Phase 4. See Section 4.2.3.

**EXCESSIVE DUP BLKS I=I (CONTINUE)**

There is more than a tolerable (usually 10) of blocks claimed by other i-nodes. See Section 4.2.3.

Possible responses to the **CONTINUE** prompt are:

YES Ignore the rest of the blocks in this i-node and continue checking with the next i-node in the file system. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system.

NO Terminate the program.

**DUP TABLE OVERFLOW (CONTINUE)**

An internal table in *fsck* containing duplicate block numbers has no more room. recompile *fsck* with a large value of **DUPTBLSIZE**.

Possible responses to the **CONTINUE** prompt are:

YES Continue with the program. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If another duplicate block is found, this error condition will repeat.

NO Terminate the program.

**POSSIBLE FILE SIZE ERROR I=I**

The i-node I size does not match the actual number of blocks used by the i-node. This is only a warning. See Section 4.2.5.

**DIRECTORY MISALIGNED I=I**

The size of a directory i-node is not a multiple of the size of a directory entry (usually 16). This is only a warning. See Section 4.2.5.

## FSCK

### PARTIALLY ALLOCATED INODE I=I (CLEAR)

I-node I is neither allocated nor unallocated. See Section 4.2.1.

Possible responses to the **CLEAR** prompt are:

YES De-allocate i-node I by zeroing its contents.

NO Ignore this error condition.

### 4. PHASE 1B: RESCAN FOR MORE DUPS

When a duplicate block is found in the file system, the file system is rescanned to find the i-node which previously claimed that block. This section lists the error condition when the duplicate block is found.

#### B DUP I=I

I-node I contains block number B which is already claimed by another i-node. This error condition will always invoke the **BAD/DUP** error condition in Phase 2. You can determine which i-nodes have overlapping blocks by examining this error condition and the **DUP** error condition in Phase 1. See Section 4.2.3.

### 5. PHASE 2: CHECK PATH-NAMES

This phase concerns itself with removing directory entries pointing to error conditioned i-nodes from Phase 1 and Phase 1B. This section lists error conditions resulting from root i-node mode and status, directory i-node pointers in range, and directory entries pointing to bad i-nodes.

#### ROOT INODE UNALLOCATED. TERMINATING.

The root i-node (usually i-node number 2) has no allocate mode bits. This should never happen. The program will terminate. See Section 4.2.1.

#### ROOT INODE NOT DIRECTORY (FIX)

The root i-node (usually i-node number 2) is not directory i-node type. See Section 4.2.1.

Possible responses to the **FIX** prompt are:

YES Replace the root i-node's type to be a directory. If the root i-node data blocks are not directory blocks, a **VERY** large number of error conditions will be produced.

NO Terminate the program.

#### DUPS/BAD IN ROOT INODE (CONTINUE)

Phase 1 or Phase 1b have found duplicate blocks or bad blocks in the root i-node (usually i-node number 2) for the file system. See Section 4.2.3 and 4.2.4.

Possible responses to the **CONTINUE** prompt are:

YES Ignore the **DUPS/BAD** error condition in the root i-node and attempt to continue to run the file system check. If the root i-node is not correct, then this may result in a large number of other error conditions.

NO Terminate the program.

**FSCK****I OUT OF RANGE I=I NAME=F (REMOVE)**

A directory entry F has an i-node number I which is greater than the end of the i-node list. See Section 4.4.

Possible responses to the REMOVE prompt are:

YES The directory entry F is removed.

NO Ignore this error condition.

**UNALLOCATED I=I OWNER=O MODE=M SIZE=S MTIME=T NAME=F (REMOVE)**

A directory entry F has an i-node I without allocate mode bits. The owner O, mode M, size S, modify time T, and file name F are printed. See Section 4.4

Possible responses to the REMOVE prompt are:

YES The directory entry F is removed.

NO Ignore this error condition.

**DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (REMOVE)**

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry F, directory i-node I. The owner O, mode M, size S, modify time T, and directory name F are printed. See Section 4.2.3 and 4.2.4.

Possible responses to the REMOVE prompt are:

YES The directory entry F is removed.

NO Ignore this error condition.

**DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T FILE=F (REMOVE)**

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry F, i-node I. The owner O, mode M, size S, modify time T, and file name F are printed. See Section 4.2.3 and 4.2.4.

Possible responses to the REMOVE prompt are:

YES The directory entry F is removed.

NO Ignore this error condition.

## 6. **PHASE 3: CHECK CONNECTIVITY**

This phase concerns itself with the directory connectivity seen in Phase 2. This section lists error conditions resulting from unreferenced directories, and missing or full *lost+found* directories.

**UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)**

The directory i-node I was not connected to a directory entry when the file system was traversed. The owner O, mode M, size S, and modify time T of directory i-node I are printed. See Section 4.4 and 4.2.2.

Possible responses to the RECONNECT prompt are:

YES Reconnect directory i-node I to the file system in the directory for list files (usually *lost+found*). This may invoke the *lost+found* error condition in Phase 3 if there are problems connecting directory i-node I to *lost+found*. This may invoke the CONNECTED error condition in Phase 3 if the link was successful.

NO Ignore this error condition. This will always invoke the UNREF error condition in Phase 4.

## FSCK

### SORRY. NO *lost+found* DIRECTORY

There is no *lost+found* directory in the root directory of the file system; *fsck* ignores the requests to link a directory in *lost+found*. This will always invoke the **UNREF** error condition in Phase 4. Check access modes of *lost+found*. See *fsck(1M)* manual entry for further detail.

### SORRY. NO SPACE IN *lost+found* DIRECTORY

There is no space to add another entry to the *lost+found* directory in the root directory of the file system; *fsck* ignores the request to link a directory in *lost+found*. This will always invoke the **UNREF** error condition in Phase 4. Clean out unnecessary entries in *lost+found* or make *lost+found* larger. See *fsck(1M)* manual entry for further detail.

### DIR I=I1 CONNECTED. PARENT WAS I=12

This is an advisory message indicating a directory i-node I1 was successfully connected to the *lost+found* directory. The parent i-node I2 of the directory i-node I1 is replaced by the i-node number of the *lost+found* directory. See Section 4.4 and 4.2.2.

## 7. PHASE 4: CHECK REFERENCE COUNTS

This phase concerns itself with the link count information seen in Phase 2 and Phase 3. This section lists error conditions resulting from unreferenced files, missing or full *lost+found* directory, incorrect link counts for files, directories, or special files, unreferenced files and directories, bad and duplicate blocks in files and directories, and incorrect total free-i-node counts.

### UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)

I-node I was not connected to a directory entry when the file system was traversed. The owner O, mode M, size S, and modify time T of i-node I are printed. See Section 4.2.2.

Possible responses to the RECONNECT prompt are:

YES Reconnect i-node I to the file system in the directory for lost files (usually *lost+found*). This may invoke the *lost+found* error condition in Phase 4 if there are problems connecting i-node I to *lost+found*.

No Ignore this error condition. This will always invoke the **CLEAR** error condition in Phase 4.

### SORRY.NO *lost+found* DIRECTORY

There is no *lost+found* directory in the root directory of the file system; *fsck* ignores the request to link a file in *lost+found*. This will always invoke the **CLEAR** error condition in Phase 4. Check access modes *lost+found*.

### SORRY. NO SPACE IN *lost+found* DIRECTORY

There is no space to add another entry to the *lost+found* directory in the root directory of the file system; *fsck* ignores the request to link a file in *lost+found*. This will always invoke the **CLEAR** error condition in Phase 4. Check size and contents of *lost+found*.

**FSCK****(CLEAR)**

The i-node mentioned in the immediately previous error condition can not be reconnected. See Section 4.2.2.

Possible responses to the **CLEAR** prompt are:

**YES** De-allocate the i-node mentioned in the immediately previous error condition by zeroing its contents.

**NO** Ignore this error condition.

**LINK COUNT FILE I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST)**

The link count for i-node I which is a file, is X but should be Y. The owner O, mode M, size S, and modify time T are printed. See Section 4.2.2.

Possible responses to the **ADJUST** prompt are:

**YES** Replace the link count of file i-node I with Y.

**NO** Ignore this error condition.

**LINK COUNT DIR I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BY Y (ADJUST)**

The link count for i-node I which is a directory, is X but should be Y. The owner O, mode M, size S, and modify time T of directory i-node I are printed. See Section 4.2.2.

Possible responses to the **ADJUST** prompt are:

**YES** Replace the link count of directory i-node I with Y.

**NO** Ignore this error condition.

**LINK COUNT F I=I OWNER=O MODE==M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST)**

The link count for F i-node I is X but should be Y. The name F, owner O, mode M, size S, and modify time T are printed. See Section 4.2.2.

Possible responses to the **ADJUST** prompt are:

**YES** Replace the link count of i-node I with Y.

**NO** Ignore this error condition.

**UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)**

I-node I which is a file was not connected to a directory entry when the file system was traversed. The owner O, mode M, size S, and modify time T of i-node I are printed. See Section 4.2.2 and 4.4.

Possible responses to the **CLEAR** prompt are:

**YES** De-allocate i-node I by zeroing its contents.

**NO** Ignore this error condition.

**UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)**

I-node I which is a directory, was not connected to a directory entry when the file system was traversed. The owner O, mode M, size S, and modify time T of i-node I are printed. See Section 4.2.2 and 4.4.

Possible responses to the **CLEAR** prompt are:

**YES** De-allocate i-node I by zeroing its contents.

**NO** Ignore this error condition.

## FSCK

### BAD/DUP FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)

Phase 1 or Phase 1b have found duplicate blocks associated with file i-node I. The owner O, mode M, size S, and modify time T of i-node I are printed. See Section 4.2.3 and 4.2.4.

Possible responses to the **CLEAR** prompt are:

YES De-allocate i-node I by zeroing its contents.

NO Ignore this error condition.

### BAD/DUP DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory i-node I. The owner O, mode M, size S, and modify time T of i-node I are printed. See Section 4.2.3 and 4.2.4.

Possible responses to the **CLEAR** prompt are:

YES De-allocate i-node I by zeroing its contents.

NO Ignore this error condition.

### FREE INODE COUNT WRONG IN SUPERBLK (FIX)

The actual count of free i-nodes does not match the count in the superblock of the file system. See Section 4.1.4.

Possible responses to the **FIX** prompt are:

YES Replace the count in the superblock by the actual count.

NO Ignore this error condition.

## 8. PHASE 5: CHECK FREE LIST

This phase concerns itself with the free-block list. This section lists error conditions resulting from bad blocks in the free-block list, bad free-blocks count, duplicate blocks in the free-block list, unused blocks from the file system not in the free-block list, and the total free-block count incorrect.

### EXCESSIVE BAD BLKS IN FREE LIST (CONTINUE)

The free-block list contains more than a tolerable number (usually 10) of blocks with a value less than the first data block in the file system or greater than the last block in the file system. See Section 4.1.2 and 4.2.4.

Possible responses to the **CONTINUE** prompt are:

YES Ignore the rest of the free-block list and continue the execution of *fsck*. This error condition will always invoke the **BAD BLKS IN FREE LIST** error condition in Phase 5.

NO Terminate the program

### EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE)

The free-block list contains more than a tolerable number (usually 10) of blocks claimed by i-nodes or earlier parts of the free-block system. See Section 4.1.2 and 4.2.3.

Possible responses to the **CONTINUE** prompt are:

YES Ignore the rest of the free-block list and continue the execution of *fsck*. This error condition will always invoke the **BAD BLKS IN FREE LIST** error condition in Phase 5.

NO Terminate the program

**FSCK****BAD FREEBLK COUNT**

The count of free blocks in a free-list block is greater than 50 or less than zero. This error condition will always invoke the **BAD FREE LIST** condition in Phase 5. See section 4.1.2.

**X BAD BLKS IN FREE LIST**

X blocks in the free-block list have a block number lower than the first data block in the file system or greater than the last block in the file system. This error condition will always invoke the **BAD FREE LIST** condition in Phase 5. See Section 4.1.2 and 4.2.4.

**X DUP BLKS IN FREE LIST**

X blocks claimed by i-nodes or earlier parts of the free-list block were found in the free-block list. This error condition will always invoke the **BAD FREE LIST** condition in Phase 5. See Section 4.1.2 and 4.2.3.

**X BLK(S) MISSING**

X blocks unused by the file system were not found in the free-block list. This error condition will always invoke the **BAD FREE LIST** condition in Phase 5. See Section 4.1.2.

**FREE BLK COUNT WRONG IN SUPERBLOCK (FIX)**

The actual count of free blocks does not match the count in the superblock of the file system. See Section 4.1.3.

Possible responses to the **FIX** prompt are:

YES Replace the count in the superblock by the actual count.

NO Ignore this error condition.

**BAD FREE LIST (SALVAGE)**

Phase 5 has found bad blocks in the free-block list, duplicate blocks in the free-block list, or blocks missing from the file system. See Section 4.1.2, 4.2.3, and 4.2.4.

Possible responses to the **SALVAGE** prompt are:

YES Replace the actual free-block list with a new free-block list. The new free-block list will be ordered to reduce time spent by the disc to rotate into position.

NO Ignore this error condition

**9. PHASE 6: SALVAGE FREE LIST**

This phase concerns itself with the free-block list reconstruction. This section lists error conditions resulting from the blocks-to-skip and blocks-per-cylinder values.

**Default free-block list spacing assumed**

This is an advisory message indicating the blocks-to-skip is greater than the blocks-per-cylinder, the blocks-to-skip is less than one, the blocks-per-cylinder is less than one, or the blocks-per-cylinder is greater than 500. The default values of 9 blocks-to-skip and 400 blocks-per-cylinder are used. See the *fsck(1M)* manual entry for further detail.

**10. CLEANUP**

Once a file system has been checked, a few cleanup functions are performed. This section lists advisory messages about the file system and modify status of the file system.

**FSCK****X files Y blocks Z free**

This is an advisory message indicating that the file system checked contained X files using Y blocks leaving Z blocks free in the file system.

**\*\*\*\*\* BOOT UNIX (NO SYNC:) \*\*\*\*\***

This message is an advisory message indicating that a mounted file system or the root file system has been modified by *fsck*. If UNIX/TS is not rebooted immediately, the work done by *fsck* may be undone by the in-core copies of tables UNIX/TS keeps.

**\*\*\*\*\* FILE SYSTEM WAS MODIFIED \*\*\*\*\***

This is an advisory message indicating that the current file system was modified by *fsck*. If this file system is mounted or is the current root file system, *fsck* should be halted and UNIX/TS rebooted. If UNIX/TS is not rebooted immediately, the work done by *fsck* may be undone by the in-core copies of tables UNIX/TS keeps.

**FSCK****INDEX OF MESSAGES**  
(Alphabetically within each section)**INITIALIZATION**

Bad -t option .....  
C<< C option? .....  
CAN NOT READ: BLK B (CONTINUE) .....  
CAN NOT SEEK: BLK B (CONTINUE) .....  
CAN NOT WRITE: BLK B (CONTINUE) .....  
Can't create F .....  
Can't get memory .....  
Can't open checklist file: F .....  
Can't open F .....  
Can't stat F .....  
Can't stat root .....  
F<< F is not a block or character device .....  
Incompatible options: -n and -s .....  
Invalid -s argument, defaults assumed .....  
Size check: fsize X isize Y .....

**PHASE 1: CHECK BLOCKS AND SIZES**

B<< B BAD I=I .....  
B<< B DUP I=I .....  
DIRECTORY MISALIGNED I=I .....  
DUP TABLE OVERFLOW (CONTINUE) .....  
EXCESSIVE BAD BLKS I=I (CONTINUE) .....  
EXCESSIVE DUP BLKS I=I (CONTINUE) .....  
LINK COUNT TABLE OVERFLOW (CONTINUE) .....  
PARTIALLY ALLOCATED INODE I=I (CLEAR) .....  
POSSIBLE FILE SIZE ERROR I=I .....  
UNKNOWN FILE TYPE I=I (CLEAR) .....

**PHASE 1B: RESCAN FOR MORE DUPS**

B<< B DUP I=I .....

**PHASE 2: CHECK PATH-NAMES**

DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (REMOVE) .....  
DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T FILE=F (REMOVE) .....  
DUPS/BAD IN ROOT INODE (CONTINUE) .....  
I OUT OF RANGE I=I NAME=F (REMOVE) .....  
ROOT INODE NOT DIRECTORY (FIX) .....  
ROOT INODE UNALLOCATED TERMINATING .....  
UNALLOCATED I=I OWNER=O MODE=M SIZE=S MTIME=T NAME=F (REMOVE) .....

**PHASE 3: CHECK CONNECTIVITY**

DIR I=I1 CONNECTED PARENT WAS I=I2 .....  
SORRY, NO SPACE IN lost+found DIRECTORY .....  
SORRY, NO lost+found DIRECTORY .....  
UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT) .....

**FSCK****PHASE 4: CHECK REFERENCE COUNTS**

BAD/DUP DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR) .....  
BAD/DUP FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR) .....  
(CLEAR) .....  
FREE INODE COUNT WRONG IN SUPERBLK (FIX) .....  
LINK COUNT DIR I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD  
BE Y (ADJUST) .....  
LINK COUNT FILE I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X  
SHOULD BE Y (ADJUST) .....  
LINK COUNT F I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE  
Y (ADJUST) .....  
SORRY. NO SPACE IN lost+found DIRECTORY .....  
SORRY. NO lost+found DIRECTORY .....  
UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR) .....  
UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR) .....  
UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT) .....

**PHASE 5: CHECK FREE LIST**

BAD FREE LIST (SALVAGE) .....  
EXCESSIVE BAD BLKS IN FREE LIST (CONTINUE) .....  
EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE) .....  
FREE BLK COUNT WRONG IN SUPERBLOCK (FIX) .....  
X<< X BAD BLKS IN FREE LIST .....  
X<< X BLK(S) MISSING .....  
X<< X DUP BLKS IN FREE LIST .....

**PHASE 6: SALVAGE FREE LIST**

Default free-block list spacing assumed .....

**CLEANUP**

\*\*\*\*\* BOOT UNIX (NO SYNC) \*\*\*\*\* .....  
\*\*\*\*\* FILE SYSTEM WAS MODIFIED \*\*\*\*\* .....  
X<< X files Y blocks Z free .....

